Probabilistic Behaviour Model and Imitation Learning Algorithm for Believable Characters in Video Games

Fabien Tencé

université de bretagne occidentale

Thèse / Université de Bretagne occidentale
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
Docteur de l'Université de Bretagne occidentale
Mention : Informatique
École doctorale Sicma

Probabilistic Behaviour Model and Imitation Learning Algorithm for Believable Characters in Video Games

présentée par
Fabien Tencé
préparée au Centre Européen
de Réalité Virtuelle
Laboratoire LISyC

Thèse soutenue le 10 novembre 2011
devant le jury composé de :

Thierry Artières (rapporteur)
Professeur des Universités, Paris 6

Yves Duthen (rapporteur)
Professeur des Universités, Toulouse 1

Pierre Bessière (examinateur)
Directeur de Recherche, CNRS

Vincent Rodin (examinateur)
Professeur des Universités, UBO

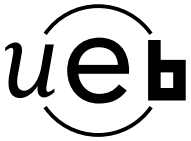
Pierre De Loor (directeur)
Professeur des Universités, ENIB

Cédric Buche (encadrant)
Maître de Conférence, ENIB

Olivier Marc (invité)
Cogérant de Virtualys

Acknowledgement

The following companies, organisms and schools made this thesis possible:



I wish to express my gratitude to Laurent Gaubert, Cédric Buche and Pierre De Loor for their assistance, support and guidance. I am also grateful to Marjorie Nicolas and Olivier Marc for giving me great freedom of choice. Special thanks to the workers at CERV, Virtualys and Cervval for their welcome and their assistance. Finally I wish to express my love to my family and Lucie for their support and understanding.

Abstract

This work aims at designing a behaviour model for the control of believable characters in video games. The character is controlled by a computer program we call an agent. We define a believable agent as a computer program able to control a virtual body in a virtual environment so that other human users in the environment think the virtual body is controlled by a human user.

As more precise criteria are needed for the evaluation of the results, we define 10 requirements for a character to be believable, based on previous experiments and several work. First the agent must react to the environment and the other players in a coherent way. This reaction must simulate a reaction time similar to human reaction time. The agent must also avoid repetitiveness, both in the actions and in the behaviour. It is also necessary that the intention of the agent can be easily understood by the human players. Contrary to what is done in most video games, the perception abilities of the agent must be similar to those of a player. The agent should be able to handle the flow of time, remembering information from the past and thinking ahead, making plans. Finally the agent has to be able to evolve, changing its behaviour for a more efficient and believable one. This evolution must be fast enough for the players to notice it, making them feel they play against an evolved being.

In order to fulfil these requirements, we studied the existing behaviour models developed both in the research and the industry. We grouped them into four types: connectionist models, state transition systems, production systems and probabilistic models. Connectionist models are very good at learning by usually have problems to handle memory and planning. State transition systems can be easily understood and modified but may are not very well adapted to learning and planning. Production systems are quite good for learning, planning and memory but make the agent act in a predictable manner. Finally, probabilistic models are good at variability, learning and memory but may show problems with planning.

As one of the requirements is that the model is able to evolve, we had to find adapted learning algorithms. In order to achieve behaviour believability, the

best method we found is imitation learning: the agent learns its behaviour using observations of one or several players. According to our definition of believability, it is the best way for the agent to look like players. Indeed, the goal of imitation learning is to make the agent act as human players. This learning method is also much faster than trial and error.

With these studies in mind we found out that the behaviour model developed by Le Hy in his thesis (Le Hy, 2007, in French) answers to most of the requirements. Le Hy's study presents a probabilistic model based on an Input-Output Hidden Markov Model (IOHMM): a hidden state is chosen according to inputs and the previous hidden state, and outputs are chosen according to inputs and the current hidden state. In Le Hy's behaviour model, hidden states are decisions, inputs are stimuli and outputs are actions. Several learning algorithms have been developed for this model in (Le Hy, 2007, in French). The best combination is a Laplace's rule for the learning of the action distributions and an Expectation-Maximization algorithm (EM) for the learning of the Markovian distributions.

However, Le Hy's model does not fulfil all the requirements. First, the number of parameters can be reduced to speed up the learning process. Indeed, more parameters means more possibilities and the learning algorithm could try to update parameters which are useless. Then the methods used to break the complexity of the distributions can make the agent behave too erratically to make the behaviour human-like. We also need the model to be able to easily adapt itself to unknown environments. In Le Hy's model, it is up to the character designers to adapt the model to the environment. Finally Le Hy's learning algorithms use simplification assumptions which can reduce dramatically the ability of the agent to evolve.

In this manuscript we propose four modifications or replacements to parts of Le Hy's model. First, we try to reduce the number of parameters in the model with a semantic refinement. Then we replace the two mechanisms to break the complexity of the probability distributions by an attention selection mechanism. We add to the model the ability to learn by imitation the layout of environments. Finally we totally revamp the learning algorithm to make it more powerful.

In order to ensure that the learning is as fast as possible, we add some *a priori* knowledge in the model by refining semantically the stimuli and the actions. This refinement comes with independence between random variables which lessen the number of parameters in the model. We introduce the notion of global and inaccurate stimuli for the choice of decisions and actions affecting

only the agent, and accurate stimuli for the choice of actions which allow the agent to interact with its environment.

As an environment is perceived through many sensors by the agent, we have to find a way to break the complexity of the combinatorial explosion of the sensors. We replace previous mechanisms developed by Le Hy by an attention selection mechanism: at each time step the agent focuses on one global and inaccurate stimulus and on one accurate stimulus and then acts as there were only these stimuli. In order to choose these stimuli, the model uses probability distributions which are expressed with functions. These functions associate for each stimulus the attention value: the higher the value the more chances the agent has to focus on the stimulus, if encountered.

For the agent to adapt to unknown environments, we enhanced and added a model named Growing Neural Gas (GNG) to the behaviour model. The GNG is a model which aims at representing volumes with a graph by learning from example coordinates in this volume. In our case, the examples are one or several players' positions. The GNG can then learn a representation of the volume of the environment which also correspond to how the players use the environment. The final representation used by the behaviour model is only composed of the graph nodes: each node represents a place where the agent can go but the edges do not give any information on the accessibility of a node from an other.

Finally, to find the parameters of our model we applied an EM method: a player is monitored during its play and all its stimuli and actions are given to the learning algorithm. The probability distributions are updated according to these observations. As our version does not make simplistic assumptions, it uses more computing power. This is counterbalanced by the reduction of the number of parameters in the model, consequence of the semantic refinement. The main downside of the new algorithm is that it is not capable yet of learning the values of the attention functions.

Each of the proposition in this manuscript has been evaluated to assess their impact on the model and on the resulting behaviour. Unfortunately, as the resulting behaviour is not believable enough to fool a player for long, we decided not to make a thorough evaluation of the believability with a sufficient number of humans players to judge the agent. However we used our knowledge of video games to point out the limitations of the models.

The semantic refinement allowed a significant decrease in number of parameters. In our application, the proposition allowed to divide the number of parameters by 10^5. Moreover, this proposition does not reduce too much the

model expressiveness as accurate stimuli are only needed for accurate actions. It is also important to notice that this semantic refinement still allows our model to be equivalent to Le Hy's model. Indeed, if the global and inaccurate stimuli and the accurate ones are the same, the two models are strictly equivalent. Therefore, our model is a generalization of Le Hy's model.

The attention selection mechanism makes the agent focus on one stimulus and act as if there were only one. It fixes the issue of the previous mechanisms aiming at breaking the complexity of the distributions Le Hy used: the agent does not switch constantly between stimuli and thus the behaviour is more understandable for the players. This fulfils one of the requirements for believability.

The GNG behaves very well with the input positions from players. We present studies of each of the GNG parameters and explain their impact. It is possible to adjust the level of detail of the graph making the nodes much closer and numerous or, at the contrary, reduce their number. The model converges rapidly to a solution, in about 10 minutes for a simple environment and 25 minutes for complex one. It is possible to increase the speed of convergence by using more than one player as an example, dividing the time by two in the better cases. Finally, the model can be left to learn for a very long period of time without fear of it to grow indefinitely. It is even recommended to let the model learn for it to be able to learn new places in the environment if players begin to visit them later in the game.

The last proposition, the new EM, is the one with the most impact on the final result. Indeed, the behaviour of the agent depends mostly on the model parameters which are learnt by the algorithm. In our application the learning algorithm converges rapidly and can handle all the observations of one player without delay. Most of the distributions converge well, with the exception, of course, of the attention functions which are not updated. When the resulting distributions are studied, the results are mitigated: the learning produce distributions which correspond to coherent stimulus-action associations but also some incoherent associations which break the illusion of believability. According to our studies, our results are more believable than our implementation of Le Hy's model but are too far from the players to hope having good believability results to a true evaluation.

Contents

List of Figures

List of Tables

List of Acronyms

FSM Finite State Machine

IP Inverse Programming

FEC Fusion by Enhanced Coherence

HMM Hidden Markov Model

IOHMM Input-Output Hidden Markov Model

GNG Growing Neural Gas

BW Baum-Welch algorithm

IBW Incremental Baum-Welch algorithm

EM Expectation-Maximization algorithm

DBN Dynamic Bayesian Network

UT2004 Unreal Tournament 2004

FPS First Person Shooter

ECA Embodied Conversational Agents

FOV Field Of View

UU Unreal Unit

EMD Earth Mover's Distance

MDS Multidimensional Scaling

List of Notations

Growing Neural Gas (GNG)	
η	Number of inputs for the classic GNG to add a new node
$\overline{Err}$	Amount of error removed to each node at each iteration
$\overline{Err}$	Maximum error for a node
$\overline{Age}$	Maximum age for an edge
Behaviour Models	
H_i^t	Random variable giving the value of the i^{th} high-level stimuli at t
N_H	Number of high-level stimuli
$\mathcal{H}_i$	Set of values the i^{th} high-level stimuli can take
H^t	Conjunction of all the high-level stimuli at time t: $H^t = (H_1^t, \ldots, H_{N_H}^t)$
$\mathcal{H}$	Set of possible values for H^t: $\mathcal{H} = \mathcal{H}_1 \times \ldots \times \mathcal{H}_{N_H}$
L_i^t	Random variable giving the value of the i^{th} low-level stimuli at t
N_L	Number of low-level stimuli
$\mathcal{L}_i$	Set of values the i^{th} low-level stimuli can take
L^t	Conjunction of all the low-level stimuli at time t: $L^t = (L_1^t, \ldots, L_{N_L}^t)$
$\mathcal{L}$	Set of possible values for L^t: $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_{N_L}$
K^t	Random variable giving the set of indexes of known low-level stimuli at time t
I^t	Random variable giving the index of the high-level stimuli the agent focus on
J^t	Random variable giving the index of the low-level stimuli the agent focus on
A_i^t	Random variable giving the i^{th} action at time t
N_A	Number of actions
$\mathcal{A}_i$	Set of values the i^{th} action can take
A^t	Conjunction of all the actions at time t: $A^t = (A_1^t, \ldots, A_{N_A}^t)$
$\mathcal{A}$	Set of possible values for A^t: $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_{N_A}$
R_i^t	Random variable giving the i^{th} reflexive action at time t
N_R	Number of reflexive actions
$\mathcal{R}_i$	Set of values the i^{th} reflexive action can take
R^t	Conjunction of all the reflexive actions at time t: $R^t = (R_1^t, \ldots, R_{N_R}^t)$
$\mathcal{R}$	Set of possible values for R^t: $\mathcal{R} = \mathcal{R}_1 \times \ldots \times \mathcal{R}_{N_R}$
E_i^t	Random variable giving the i^{th} external action
N_E	Number of external actions
$\mathcal{E}_i$	Set of values the i^{th} external action can take

E^t	Conjunction of all the external action at time t: $E^t = (E_1^t, \ldots, E_{N_E}^t)$
$\mathcal{E}$	Set of possible values for E^t: $\mathcal{E} = \mathcal{E}_1 \times \ldots \times \mathcal{E}_{N_E}$
D^t	Random variable giving the decision at t
$\mathcal{D}$	Set of possible values for the decision
$\mathcal{A}^{t_0..t_1}$	Actions between t_0 and t_1. For Chameleon $\mathcal{A}^{t_0..t_1} = (E^t, R^t)_{t \in [\![t_0, t_1]\!]}$
$\mathcal{S}^{t_0..t_1}$	Stimuli between t_0 and t_1: $(L^t, H^t, K^t)_{t \in [\![t_0, t_1]\!]}$
$\mathcal{Q}^{t_0..t_1}$	Hidden state between t_0 and t_1: $(I^t, J^t, D^t)_{t \in [\![t_0, t_1]\!]}$
	Expectation-Maximization algorithm (EM)
Φ	Parameters of the model
Φ_n	Parameters of the model at the nth iteration of the EM
T	Length of the sequence of observation used for the learning
$\mathcal{A}$	Whole sequence of observed actions: $\mathcal{A}^{1..T}$
$\mathcal{S}$	Whole sequence of observed stimuli: $\mathcal{S}^{1..T}$
$\mathcal{Q}$	Whole sequence of hidden states: $\mathcal{Q}^{1..T}$
α	Forward variable (estimator)
β	Backward variable (estimator)
γ_t	Estimator of the model to be in hidden state given the observations
ξ^t	Estimator of the model to do a transition between two hidden states given the observations

Contents of Chapter 1

Chapter 1

Introduction

Summary of 1

In this chapter we explain why increasing the believability of computer-controlled characters in a virtual environment can make the users forget they are in a simulation. As believable computer-controlled characters are very important in video games, we choose to focus on this domain for our research. We list 10 requirements for such characters to be believable. They must: react to events, simulate reaction time, have variations in the movements, surprise the player with unpredictability, evolve, evolve fast enough to be observable, be able to plan, have an understandable behaviour, simulate a human-like perception and be able to memorize events. We present briefly our study and the outline of this manuscript.

Introduction of 1

Imagine ourself in the middle of a naval battle, giving orders and fighting corsairs, or imagine you roam in a crowded souq on an alien planet, stallholders trying to sell you their high-tech merchandise. These scenes look like they are taken from films but the whole environment, the people feel real, they react to you as they were alive.

Of course you cannot be in such a place, but virtual reality can make you feel like you are in those incredible situations. The main challenge for the designers of virtual reality systems is to make you forget that you are in a simulation. This is specially true for the characters populating the virtual world which must give you the feeling of being alive.

The goal of this thesis is to make it possible to design computer controlled characters which gives this feeling of vividness. This kind of character can be very useful to replace humans in virtual environments where they lack. They can fill a whole world with life-like characters, making the experience richer and more interesting.

Plan of 1

We first explain why we focus on the believability of video games characters in section 1.1. Then we define the criteria with which we evaluate such characters throughout the manuscript in section 1.2. We explain how our work may be a solution to this problem in section 1.3 and detail the plan of this manuscript in section 1.4.

1.1 From Virtual Reality to Believable Agents in Video Games

In this thesis, we focus on virtual reality systems. They allow both humans and computer programs to interact in a virtual environment which consists in images, sounds, etc. often depicting a place. All these stimuli are produced by computer programs and electronic devices for users to interact with and in this fictive place.

1.1.1 On the Need of Believable Agents in Virtual Reality

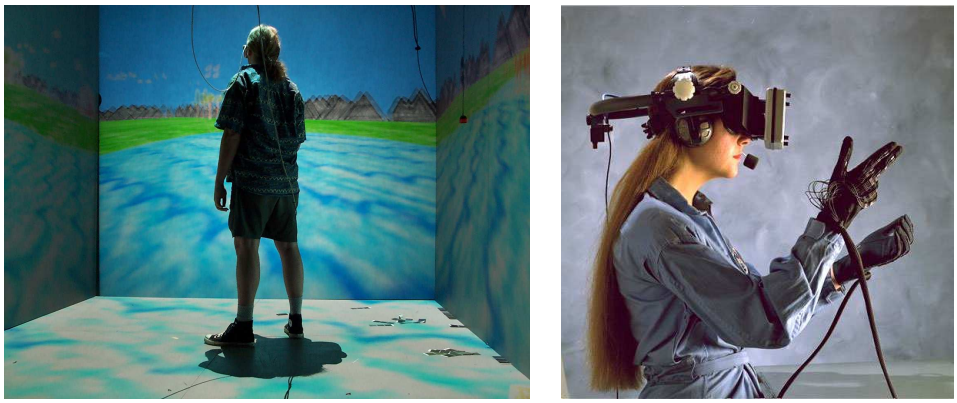
The goal of virtual reality is, by definition, to substitute a user's physical reality by a virtual, computer-generated, one (Burdea and Coiffet, 2003). In an attempt to quantify how well this substitution is achieved, two criteria have been defined in academic research: *immersion* and *presence* (Slater et al., 1996). Their exact definitions are a subject of debate among the researchers, we will present the most widely used ones.

The first criterion, immersion, is an objective criterion which is influenced by the hardware and software (Slater et al., 1995). It depends on the virtual sensory information's types, variety, richness, direction and to which extent they override real ones. For instance, force feedback and motion sensing controllers, surround sound and high dynamic range rendering can improve the immersion (see figure 1.1).

The second criterion, presence (Heeter, 1992), also called telepresence (Steuer, 1992), is a more subjective criterion. It is defined as the psychological sense of "being there" in the environment (Gibson, 1986) (see figure 1.2 for an example). Presence partly depends on the match between sensory data and internal information (Loomis, 1992; Slater et al., 1995). Part of this internal information consists in personal and subjective models of the world. These models allow a deeper understanding of the perceptions and a better anticipation of future events (Held and Durlach, 1991).

This comparison between what we observe and our understandings of the environment is close to what is called *believability* in the arts (Riedl and Young, 2005): fictional objects, places, characters and stories are believable only if the

perception we have of them mostly fits what we expect, our models. It is reasonable to think that by improving the believability of a virtual environment, we will improve the presence as well.



(a) (b)

Figure 1.1: *Two widely used immersive systems: the Cave Automatic Virtual Environment (a) and the classic goggles and gloves (b). Their goals are to replace most of the user's perceptions of the real world by virtual ones and to make the interaction with the environment more natural.*

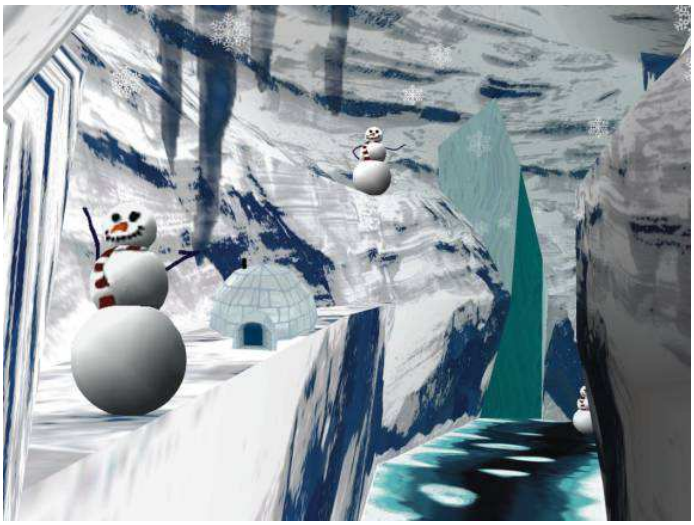


Figure 1.2: *A virtual environment depicting a world with snow and ice: Snowworld (Hoffman et al., 2003). Although the game is not very immersive, the feeling of presence can be strong: patients suffering burns reported to be less subject to pain while playing the game. The game could make its users feel cold even if this is not the case in the real world.*

Two main areas in academic research aim at improving the believability of virtual environments: interactive storytelling and believable computer-controlled characters. In interactive storytelling (Cavazza et al., 2002; Riedl and Stern, 2006; Crawford, 2004), a plot is generated to create a series of appealing events for the user. With believable computer-controlled characters (Loyall, 1997; Bates, 1992), the goal is to create vivid characters capable of interacting with both users and other characters. This study focuses on virtual environments where users can roam where they want and do as they please. In this kind of environment, storytelling is almost nonexistent. For this reason, in this thesis we focus on making believable computer-controlled characters.

Before going further, we will define some terms to avoid confusions. In a virtual environment, users and computer programs can have virtual bodies. We will call these bodies *avatars*, as they are the representations of entities in the virtual world. Avatars can be controlled by a human or by a computer program we call *agent*. We use the word agent because the program will take sensory inputs from the world and give back actions commands. Finally we will use the word *character* to designate the couple agent-avatar.

1.1.2 Believable Characters in Video Games

This thesis focuses on the design of believable characters in virtual environments. However, there are a lot of different environments. The choice of one environment has a lot of impact on the definition of the believability. Therefore we must choose a kind of environment to work with before going further.

Research on believable characters in virtual reality is often linked to video games (McMahan, 2003). The first reason is because game designers really need believable characters to give the best gaming experience to the players. The second reason is because scientists need a complete and easy-to-use environment where to develop believable characters which they can found in video games (Laird and Lent, 2001). Video games are thus a very useful platform to work on believability and to take inspiration from. We will now use the word *player* to designate a human user of a video game.

As believable characters are the very goal of our work, we need *character-centric* video games. For their behaviours to express fully, agents should be able to interact with the environment, other agents and players. The more complex and the higher the number of *interactions*, the harder the believability is to achieve. However, *speech* is a very particular form of interaction and we do not want to handle that one for now. Finally, in this thesis, we focus on open environments where the programmer cannot predict what will happen.

Such games are often called *sandbox* games. Therefore we need a sandbox video game based on characters with lots of interactions except speech.

The video game industry being very prolific, we first need to categorize video games before choosing one for our application. Laird and Lent (2001) and Mac Namee (2004) had already defined different groups:

- Action: players and agents take the role of fighters, running around and trying to kill one another.
- Role playing: players take the role of heroes in fantastic or futuristic environments. They must use diplomacy and strength to make their way to victory.
- Adventure: players follow a plot and solve puzzles and riddles.
- Strategy: players command armed units to fight battles, build bases and manage resources.
- God games: players have god-like control over the environment to build cities, shape the landscape and rule over populations.
- Team sports: players and agents play coach and players in a team sport like football, basketball, hockey, ...
- Individual sports: players and agents play individual sports such as golf, skate, racing, ...

	Character-centric	Interactions	Speech	Sandbox
Action	✓	✓	✗	✓
Role playing	✓	✓	✓	✓
Adventure	✓	✗	✓	✓
Strategy	✗	✗	✗	✗
God games	✗	✗	✗	✓
Team sports	✗	✓	✗	✗
Individual sports	✓	✓	✗	✗

Table 1.1: *Characteristics of the main types of video games for the development of believable characters.*

Table 1.1 gives, for each type of video game, if they answer or not to the four criteria we defined. It shows that actions games seem to be the best choice to develop believable characters. The classic example of such action games is first person shooter games. In those games, each player or agent controls a unique avatar and sees through its eyes. The character can, non-thoroughly, grab items (weapons, ...), move (walk, jump, ...) and shoot with a weapon. Each player and agent has an amount of hit points, also known as life points: each time an avatar is hit by an enemy fire, a certain amount of hit points are subtracted to the current value. When hit points reach zero, the avatar "dies" but can usually reappear at another place in the game's environment. Although the concept can seem very basic, it can prove challenging to design believable characters. In order to make things simpler at the beginning, it is possible to avoid cooperative rules as tactics used add a lot a complexity to the learning.

1.2 Assessing the Believability of Characters in Video Games

Now that we chose to work on the believability of characters in video games, we have to explore the exact definition of *believability*.

1.2.1 Believable Characters: Two Definitions

As the notion of believability is subjective, it is very complex to define what a believable character is. In order to understand this concept, we must look at its meaning in the arts where it is a factor of *suspension of disbelief* (Bates, 1992). Suspension of disbelief is when even though a reader or a spectator knows that the story and the characters are not real, he/she may "forget" it and have feelings and reactions as if the story were true.

According to Thomas and Johnston (1981), two core animators of Disney, believable characters' goal is to provide the "illusion of life" (see figure 1.3). Riedl and Young (2005, page 2) defines with more details of how achieving this peculiar goal: "Character believability refers to the numerous elements that allow a character to achieve the 'illusion of life', including but not limited to personality, emotion, intentionality, and physiology and physiological movement". Loyall (1997, page 1) tries to be more objective stating that such a character "provides a convincing portrayal of the personality they [the spectators] expect or come to expect". As we pointed out, this definition is quite close to one factor of the presence, the match between users' models and sensory data.

Figure 1.3: *Disney made the character of Bambi very believable despite being a fictive and non-realist deer in a cartoon. It gives the impression of being alive and thus is able to provoke feelings.*

Applying the believability definition for video games is even more difficult. Unlike classic arts, players can be embodied in those environment by the mean of avatars and can interact with the characters. It is also possible that several players are embodied in the same environment, each one knowing that they might encounter intelligent and sentient entities. The believability question is now: does a believable character have to give the illusion of life or have to give the illusion that it is a player? (Livingstone, 2006; Tencé et al., 2010). There can be very important differences as players may not act as in the real world. Indeed, even if the virtual environment depicts the real world, players know that their acts have no real consequence.

In this thesis, we will consider only believable as *giving the illusion of being a player* (see figure 1.4). At first glance, it can be seen as going against presence as we remind the players that there is a real world. However, not using this approach has some drawbacks too: if the player becomes aware that there is no other intelligent and sentient being in the simulation, characters giving the illusion of life can be classified as "being a piece of program" which may break the illusion permanently. Users may also see problems in the characters' behaviour only because they know they are not human-controlled.

Now that we have defined believability, we have to find how to improve it and measure the improvement.

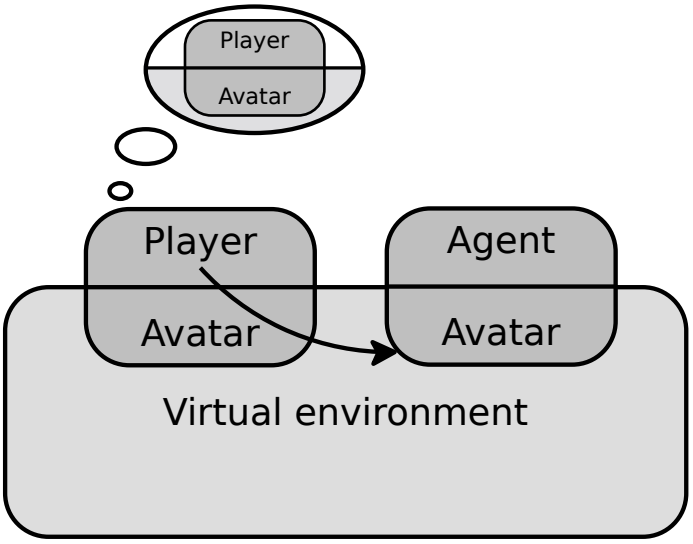


Figure 1.4: *A representation of the notion of believability chosen in this thesis. The arrow represents the player observing the avatar of the agent in the virtual environment.*

1.2.2 Believability Requirements for Agents in Video Games

As believability is a broad concept, we need to find more precise criteria to break this concept down. According to an exploratory research (see protocol and questionnaire in appendix), we listed criteria which were reported to have an impact on believability. The requirements for believability are listed in table 1.2.

Believability requirement	Summary of the requirement
[**B1: Reaction**]	React to the players and changes in the environment
[**B2: Reaction time**]	Simulate a human-like reaction time
[**B3: Variability**]	Have some variability in the actions
[**B4: Unpredictability**]	Surprise the players with unpredictable behaviour
[**B5: Understandable**]	Have a understandable behaviour
[**B6: Perception**]	Have human-like perception
[**B7: Planning**]	Plan actions in the future to avoid mistakes
[**B8: Memory**]	Memorize information
[**B9: Evolution**]	Evolve to avoid repeating mistakes
[**B10: Fast Evolution**]	Evolve fast enough for the players to see it

Table 1.2: *List of the requirements for a character to be believable.*

As we only did an exploratory research, one cannot draw conclusions about the relevance of these factors. However several work confirmed our findings.

[B1: Reaction] The most basic requirement for a believable character is to react to changes in the environment and especially to a player (Livingstone, 2006; Wetzel, 2004). Players should feel that the character is reacting to them, reinforcing the overall feeling of presence. Any character not fulfilling this requirement will break the illusion of believability very rapidly.

[B2: Reaction time] The speed at which the character reacts to changes is also quite important (Laird and Duchi, 2000; Livingstone, 2006). Instant or slow reactions may give the players clues about the real nature of the character. According to experiments, a reaction time of around 100 ms, which is a very low estimate of the human reaction time, give the best results in term of believability (Laird and Duchi, 2000).

[B3: Variability] The way the action is done have an impact on believability. Very accurate movements are known to break the illusion of believability (Laird and Duchi, 2000; Livingstone, 2006) and (Loyall, 1997, page 18). Movements should be the closest possible to what a player would do because it is the first thing one sees when observing an avatar. For example, inaccuracy should be added to the movement to create a feeling of humanness.

[B4: Unpredictability] What is true with actions is also true with the whole behaviour. Agents doing over and over the exact same thing are rapidly categorized as being artificial. Adding some unpredictability can improve a lot believability (Bryant and Miikkulainen, 2006; Isla, 2005). The difficulty is that too much unpredictability can give the feeling of randomness which can harm believability too (Isla, 2005).

[B5: Understandable] Unlike realistic characters, believable characters might be forced to overdo for observers to understand what they are doing (Pinchbeck, 2008; Isla, 2005). Although it can seem strange, it is sometimes mandatory to exaggerate some of the character's characteristics so that people believe in it. This technique is very often used in arts, especially in cartoons. This means that human-like characters could have a quite low believability. There are however links between realism and believability so it should be a good start to draw inspiration from realism. It is also important to note that believability should be easier to achieve than realism because we do not want the character to exhibit a truly human-like behaviour but only not to break the illusion to be human-like.

[B6: Perception] It is quite usual in video games to grant characters with "super-natural" perceptions to make them look like more clever than they are (Cass, 2002; Mac Namee, 2004, page 34). This technique may work quite well if used carefully, but when discovered by players it breaks definitively the illusion of believability (Laird and Lent, 2001). In order to avoid this kind of problem, it is better to give to the characters perception information which is the closest possible of the one players have.

[B7: Planning] In order to avoid making counter-productive actions which could break the illusion of believability, the character may need to have the ability to plan or anticipate. The character could thus choose actions not leading to easily avoidable mistakes (Livingstone, 2006) by predicting the outcomes of its choice. However, planning in vast and rich virtual environments is a hard problem to solve because of the combinatorial explosion of possibilities.

[B8: Memory] Surprisingly, memory has not been cited by researchers on believability as an important criterion. However, nowadays most of the models used for characters have some kind of mechanism to memorize information. It can be linked to what Loyall (1997, page 22) calls the consistency of the expression. The expression and behaviour of a character must be consistent at one moment but over the time too. For this coherence to last, the character must have some kind of memory not to forget past events and to act according to them.

[B9: Evolution] Memory and evolution can be seen as two faces of the same coin, so the character must have mechanisms to evolve for instance by learning (Loyall, 1997, page 20). A character keeping the same behaviour even if it is clearly counter-productive breaks the illusion of being human-controlled. Agents able to learn will tend to surprise the players by exhibiting new behaviour, adding a bit of unpredictability during the interaction. That is the main reasons why believable characters must be able to learn (Thurau et al., 2005; Gorman and Humphrys, 2007).

[B10: Fast Evolution] Evolving itself it not sufficient, the players must see that the character evolves. Indeed, if players do not see the character evolving, they will assume that it does not. The evolution, or the learning, must be quick enough to be observable. "Quick enough" is not easily quantifiable and may vary: avoiding big mistakes should be learned within minutes after doing the mistake whereas subtle strategy may takes hours or days to be learned.

All those criteria are to be tempered by what have been named *Eliza effect* (Weizenbaum, 1966) (see figure 1.5 for en example). This principle is central for believable characters: as long as the character is not actively destroying

the illusion, people tend to see complex thinking mechanisms where there are not. All the listed criteria are thus indications of how to make a character behave, but illusion is enough.

User:	Men are all alike.
ELIZA:	In what way?
User:	They're always bugging us about something or other.
ELIZA:	Can you think of a specific example?
User:	Well, my boyfriend made me come here.
ELIZA:	Your boyfriend made you come here
User:	He says I'm depressed much of the time.
ELIZA:	I am sorry to hear you are depressed
User:	It's true. I am unhappy
ELIZA:	Do you think coming here will help you not to be unhappy
User:	I need some help, that much seems certain.
ELIZA:	What would it mean to you if you got some help
User:	Perhaps I could learn to get along with my mother.
	...

Figure 1.5: *Example of a discussion with the chatbot ELIZA. It illustrates that ELIZA, while not being truly active in the conversation, does not break the illusion of believability. This excerpt of discussion is taken from (Weizenbaum, 1966).*

1.3 Objective of our Work

The global objective of this study is to find a way of designing a computer program which can answer to the ten requirements of believability. The goal is both to have a generic way of creating believable character and to have a real program that is giving actual results.

Design a Behaviour Model To do so, we want to find a behaviour model able to generate believable behaviours in a wide range of different virtual environments. Model allows us to abstract from the application used for the tests in this thesis. As there are many existing models for the control of avatars in virtual environment the first objective is to evaluate them given our ten requirements.

Design a Learning Algorithm We also pointed out that learning is a vital component of a believable character. We have to analyse the different solutions in the domain of machine learning and find the one that best fits the goal of believability. The main difficulty is that models and learning algorithms are often linked together so the objective is to find a couple.

Validate Results Finally we need to assess to what extent the model and its implementation achieve the goal of believability. We also need to evaluate if the model is easy to implement and to adapt to new environments.

1.4 Organization of this Manuscript

In **chapter 2** we study the different behaviour models which can best answer to the ten requirements for believability. We also study learning algorithms which would allow the model both to evolve [B9: Evolution] and to adapt to any new virtual environment. We find that the model proposed in Le Hy et al. (2004) answer well to our problem. We point out weaknesses in the behaviours produced by the model and in its designs.

In **chapter3** we propose four modifications to improve the model. We describe a semantic refinement which allow a reduction of the number of parameters and a finer control of the avatar's actions. We also modify the way the complexity of the model is handled by introducing an attention selection mechanism which allows more complex behaviours at the cost of more parameters. For the character to adapt to new environments we also detail an imitation learning algorithm able to learn the topology of the environment. Finally, we revamp the whole learning algorithm, reapplying the whole EM procedure for a finer learning of the behaviours.

In **chapter4** we explain how the model is applied to the video game we chose for the evaluation and evaluate the gains of our four propositions. We analyse the change in the number of parameters implied by the semantic refinement and the attention selection and give examples of how the modifications improve the behaviours. We also give a detailed analysis of the convergence of the imitation learning algorithms both for the environment and for the behaviour. Finally, we study the results given by the learning algorithm and see if they can make the model generate believable behaviours.

Last, in **chapter5** we conclude on how our work answered to the initial problem. Our work is however not exempt of problems because some parameters make the behaviour of the agent ill-adapted to certain situations. Therefore, we also propose several ways to improve the results and to make the model more flexible.

Contents of Chapter 2

Chapter 2

Behaviour Models and Learning Algorithms for Believable Agents

Summary of 2

This chapter presents our study of solutions for the modelling and the learning of believable behaviours for agents in video games. We analyse connectionist models, state transition systems, production systems and probabilistic models. We also study the different possibilities for the learning, finding that imitation learning should be the best choice to achieve believability. We conclude that the work in (Le Hy et al., 2004), a probabilistic model with an imitation algorithm, seems to be the better solution for our problem. However, some connectionist models could also be integrated to improve the results. In order to confirm this study, we present briefly an implementation of Le Hy's model. We conclude by pointing out several problems in both its design and results: the independence hypothesis between actions, the Fusion by Enhanced Coherence (FEC), the IP and the approximations made in the Incremental Baum-Welch algorithm (IBW) make the agent unable of having and learning complex behaviours. The model has also difficulties in adapting to new environments.

Introduction of 2

As video games often feature vast virtual environments with lots of possible interactions, it is almost impossible to predict all the possible scenarios. Therefore, we need a model to control each agent. This model have to choose the best actions to do, depending on the agent's perceptions. Researchers tried to find solutions to this problem for several decades, giving birth to an important number of models.

Plan of 2

In this chapter we will first present the four main families of behaviour models in section 2.1 and see how well they could answer to the believability criteria listed in section 1.2. We then explain why an imitation learning algorithm is essential for believability because of requirement [B9: Evolution] and we list different work on this kind of algorithm in section 2.2. We argue that Le Hy's proposal (Le Hy et al., 2004) may be the best solution for the learning and generation of believable behaviours and we explain the theory behind this work in section 2.3. We present our implementation of the model in section 2.4 and list some of the weaknesses in the model and in the resulting behaviours.

2.1 Behaviour Models for Believable Agents

Introduction 2.1

As there are very few evaluations of the believability of behaviour models, we will try to find the models which fulfil most of the requirements for believability. We must keep in mind that as long as the behaviours generated by the model are not evaluated, the model cannot be considered to be a solution to our problem. However, evaluating each model is infeasible because it would need too much time and resources.

Plan 2.1

First we will list the requirements models for believable agents must fulfil and categorize potential solutions in section 2.1.1. Then, we will analyse the four categories, connectionist models in section 2.1.2, state transition systems in section 2.1.3, production systems in section 2.1.4 and finally probabilistic models in section 2.1.5.

2.1.1 Requirements and Possible Solutions

Because of the context of this thesis, we will only look at models for embodied agents in the following study. Those models can handle interactions with virtual environments and avatars. Therefore, they all fulfil the requirement [B1: Reaction]. According to the criteria we listed in section 1.2, we list requirements for the model itself in the table 2.1.

In these criterion, we do not take into account [B6: Perception] because we consider it is not the role of the behaviour model to model human-like perception. This criterion will be useful when we will have to choose which data will be given to the model in the implementation.

As we already stated, there are a lot of available models for the control of behaviour. Instead of examining each model, we will rather study models by category. We choose to categorize behaviour models into 4 types:

- Connectionist models
- State transition systems
- Production systems
- Probabilistic models

Model requirements	Summary of the requirement
[M1: Variability]	Model variations in the actions and behaviours [B3: Variability] and [B4: Unpredictability]
[M2: Learning]	Is compatible with learning algorithms [B9: Evolution]
[M3: White box]	Can be modified and parametrized manually (Isla, 2005) to make the agent overdo [B5: Understandable]
[M4: Exaggeration]	Generate exaggerated behaviours so that players can easily understand the agent's objectives [B5: Understandable]
[M5: Planning]	Elaborate plans to avoid doing easily avoidable mistakes [B7: Planning]
[M6: Reaction time]	Simulate reaction time [B2: Reaction time]
[M7: Memory]	Model memory [B8: Memory]

Table 2.1: *Requirements for the models to make agents express believable behaviours.*

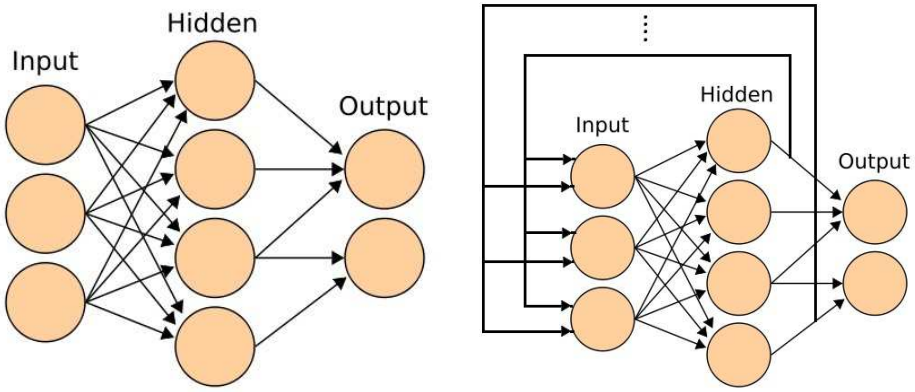
2.1.2 Connectionist Models

Connectionist models are composed of small computing units linked together to form a network. As written in (Feldman and Ballard, 1982):

> "The fundamental premise of connectionism is that individual neurons do not transmit large amounts of symbolic information. Instead they compute by being appropriately connected to large numbers of similar units."

This approach gained popularity in the mid-1980s and is still very used to model behaviours.

Usually, input units are linked to sensory data and output units to motor commands. Those two layers are rarely connected directly together, instead some hidden layers are inserted in between (see figure 2.1 for examples). The input layer is then connected to a hidden layer and the hidden layer to other hidden layers or to the output layer. The network connecting all those layers may have various forms: feedforward, recurrent, etc.



(a) (b)

Figure 2.1: *Two simplified examples of (a) a feedforward neural network and (b) a recurrent neural network. These are the most widely used connectionist models. Images taken from Wikimedia Commons.*

Connectionist models are used in some video games to model simple behaviours which may be learned during the play. For example the player may teach animals (see the games *Black and White* and *Creatures*) or the game makes the player's opponents learn driving by imitation (see the games *Colin McRae Rally 2* and *Dirt Track Racing*) (Charles and McGlinchey, 2004; Johnson and Wiles, 2001). When it comes to complex behaviours, they are only used as modules, for the modelling of specific parts of the behaviour. For instance, Gorman and Humphrys (2007) use neural networks to model aiming and firing in a 3D shooting game. The model is able to reproduce short-term anticipatory behaviours and subtle human characteristics.

The main advantage of connectionist models is the great ability to learn [M2: Learning]. Many techniques can be used to change the parameters of each computing units (Hinton, 1989) so that to optimize the results. They can be used for short-term anticipation but they do not have any inner mechanism to simulate planning [M5: Planning]. It is however possible to model reaction time with connectionist models [M6: Reaction time].

However, the main drawback is that parameters can only be learned because connectionist models are *black boxes* [M3: White box]. As there is no semantic associated to computing units and links, it is almost impossible to modify them. Memory can be also hard to model, recurrent neural networks, which try to resolve this problem can give chaotic behaviours [M7: Memory]. Finally, connectionist models do not produce inherently unpredictable [M1: Variability] or exaggerated behaviours [M4: Exaggeration].

Considering all those characteristics, a behaviour model composed of only connectionist models would have difficulties to make the illusion of believability last for long. However, those models can be used to handle part of the whole model like the realisation of actions for example. Their use should be based on the application in which the agent is used.

2.1.3 State Transition Systems

State transition systems are composed of states and transitions. Each state may be activated or not. Each transition associate one state, p, to another, q, in a directed relation $p \xrightarrow{\alpha} q$ with a label α. Transition are triggered if p is activated and under the circumstance α which can be based on inputs or outputs.

In order to generate behaviour, state transition systems use sensory data for the condition of transitions. States model either types of behaviour or actions. It results that depending on the sensory information, actions are done following the logic sequencing of behaviours-states.

State transition models are widely used in the video game industry in the form of Finite State Machines (FSMs) (Cass, 2002) (see figure 2.2) to express the whole behaviour of autonomous agents in First Person Shooter (FPS) (see the games *Quake* and *Doom*) (Johnson and Wiles, 2001). They give quite believable behaviours because character designers are quite skilled to bend them to their needs. However, as FSMs rapidly become unmanageable when complexity increases, behaviour trees are beginning to replace them because there are more flexible and easier to maintain (Dromey, 2003) (see figure 2.3). State transition models are less used in research, but there are still some studies (Donikian, 2001) for the control of agents in virtual environments which are based of it.

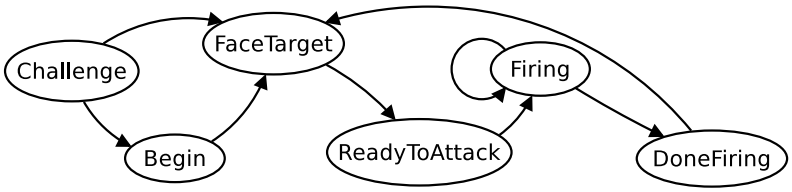


Figure 2.2: *A part of a video game agent's behaviour modelled with a FSM (Le Hy, 2007, page 27). The conditions for each transition is not given because of their complexity.*

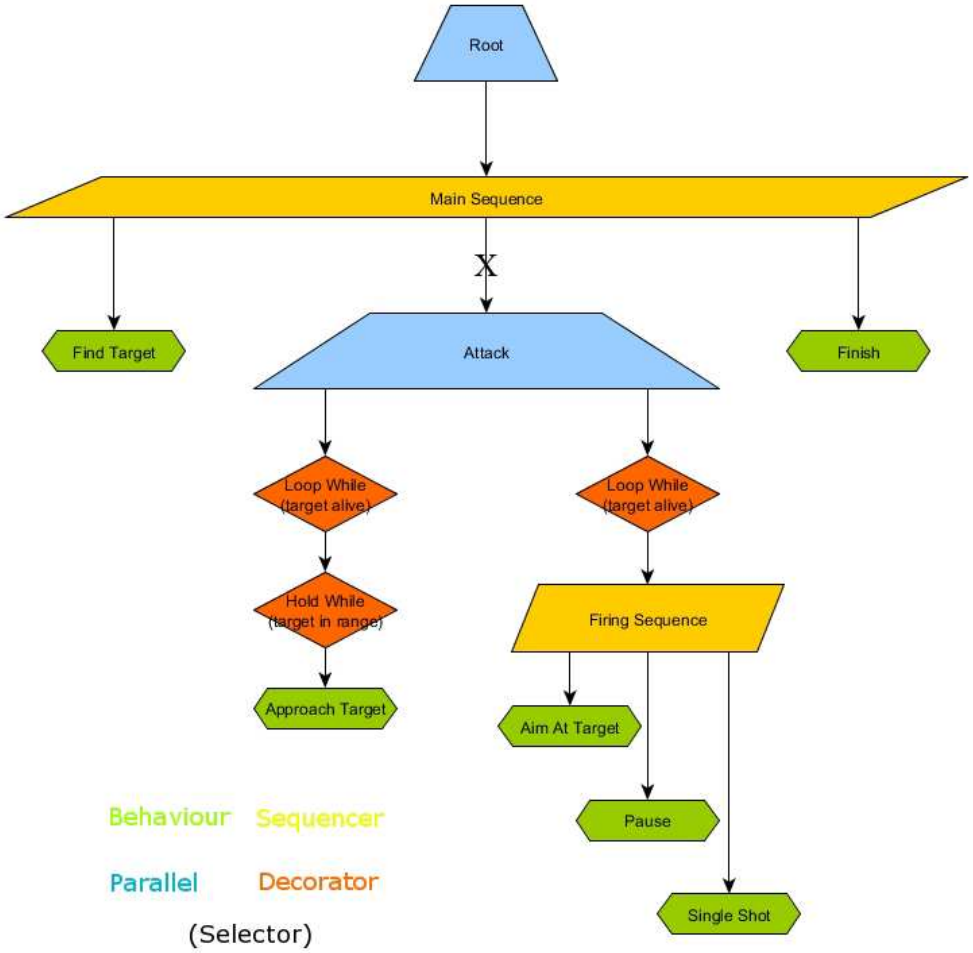


Figure 2.3: *Behaviour trees are more flexible way to express behaviour than FSMs. This behaviour tree is used to model an agent in a FPS. Taken from* `http://www.garagegames.com/community/blogs/view/18589`.

The main advantage of state transition systems are that they are easy to read and thus to modify [M3: White box]. It allows fine modifications to adjust the behaviour. As each state often represents a kind of behaviour, it is possible to recognize what the agent want to do [M4: Exaggeration]. The transition system allows the agent to follow a logical sequential behaviour which can give the illusion of memory but may be too limited for the illusion to fool the players for long [M7: Memory]. Finally, complex state transition systems can model reaction time while basic one cannot [M6: Reaction time].

The main drawback is the lack of learning algorithms developed for this kind of models [M2: Learning]. There are algorithms capable of building FSMs from data but they are designed for the learning of grammars (Angluin, 1982; Zeng et al., 1993) and not behaviours. Another drawback is that state transition systems often produce very predictable behaviours which are known to break the illusion of believability [M1: Variability]. They do not offer a planning system neither [M5: Planning] and it does not really model memory [M7: Memory].

Although FSMs and behaviour trees may gives the best example of believability it is only because character designers often have the time and knowledge to adjust the behaviours for the environment and the players. The models are often complex to maintain because of the lack of learning algorithm and the numerous interconnections. This is specially true in the kind of environments we aim at: vast worlds, with lots of interactions makes too many unpredictable situations for characters designers to anticipate them all. State transition systems may be used for simple behaviours but not for believable human-like behaviours, so this kind of models seems to be ill-adapted to our problem.

2.1.4 Production Systems

Production systems are the most used kind of behaviour model in research. Those systems are based on a set of conditional rules often in the form of *IF ... THEN ...*. Each rule may be fired (activated) when the *IF* condition is met. When the rule is fired, the *THEN* statement applies. Obviously, these rules are very simple to express in all programming languages because the conditional rules are almost present in all of them. This is one of the explanation of the popularity of such models.

In order to control a behaviour, the *IF* condition is based on sensory and internal information and the *THEN* statement activates motor commands. If several rules are fired together, some mechanisms can be used to resolve potential conflicts, like for instance in expert systems. Rules can also be chained, using rules as sets of preconditions or goals to elaborate plans.

Production systems regroup a wide range of models. The well known cognitive models Soar (Laird et al., 1986; Newell et al., 1987) and ACT-R (Anderson, 1993) (see figure 2.4) try to model all the human thinking mechanism. Classifiers systems (Sanchez et al., 2006; Sigaud and Wilson, 2007) are also used to control agents and have the advantage of being able to learn the whole behaviour (Robert and Guillot, 2005) as in the game *Ryzom*. The Oz Project was focused on making believable behaviour (Bates, 1992; Reilly, 1996; Loyall, 1997). Production systems are not only used in research: in the video game *F.E.A.R.*, which was acclaimed for the quality of its agents, the behaviour model (Orkin, 2006) is partially based on the STRIPS planner (Fikes and Nilsson, 1971). It allows agents to set up complex tactics which seem rather human-like, see Table 2.2 for an example.

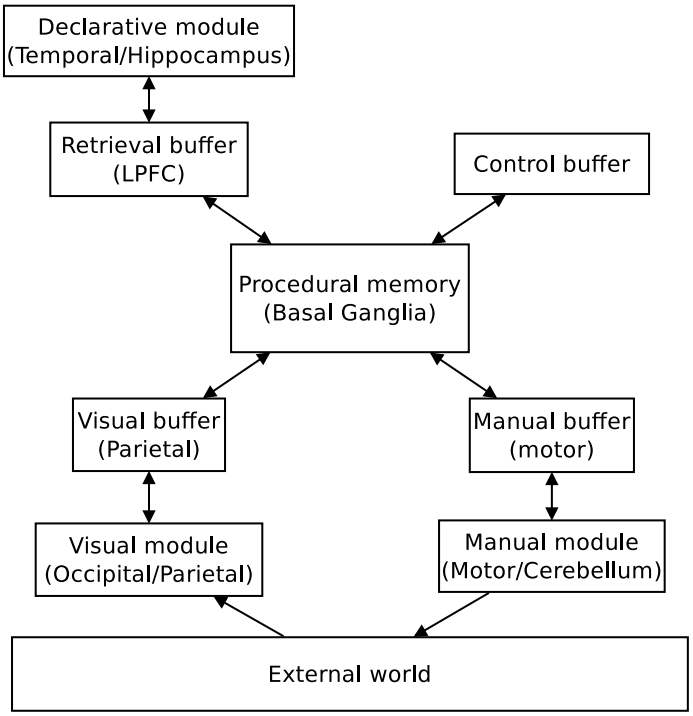


Figure 2.4: *A general view of the architecture of ACT-R (Anderson, 1993). ACT-R is a cognitive model which uses production rules to generate the behaviours.*

State	phone	recipe	hungry
Goal	—	—	No
Rule 1			
Condition	Yes	—	Yes
Action	Call pizza delivery		
Result	Yes	—	No
Rule 2			
Condition	—	Yes	Yes
Action	Make cake		
Result	—	Yes	No

Table 2.2: *A very simple example of a STRIPS planner. The world is represented by a state (is there a phone, do I have a recipe, am I hungry) and the agent have its goals represented by a state. In order to satisfy hunger, the agent can use two plans depending on its resources. Here, plans consist only on one rule, but it can be a chaining of rules, requiring a sequence of actions. Example inspired by (Orkin, 2006).*

As many behaviours can be expressed as *IF ... THEN ...* rules, production systems are well adapted to model them. This systems can model planning in a very easy and compact way [M5: Planning]. A memory can be integrated and updated by rules allowing consistent behaviour over time [M7: Memory]. It is possible to define noticeable behaviours by defining internal state selecting the appropriate set of rules [M4: Exaggeration]. Some models support well learning algorithms while other do not or have very strict constraints [M2: Learning]. Reaction time can be modelled easily [M6: Reaction time]. Finally, is often easy to understand and modify rules because they are expressed in a way people understand [M3: White box].

However, rules are binary: they fire or they do not (except when coupled with a probabilistic model like the random pick in classifiers). As a consequence, the outcome is predictable [M1: Variability]. Players could find those regularities, breaking the illusion of believability. As some production systems are aiming at realism they may be too complex to generate "only" believability. This is particularly true with cognitive models which have lots of modules, making the first approach difficult [M3: White box]. Introducing such models in the video game industry may prove to be very difficult because programmers are more used to use FSM, which are more straightforward.

Although production models seems to answer most of our needs, few model actually have all the characteristics. Because of their complexity, it is nearly impossible to merge models to combine their advantages. Nevertheless, production systems seems one of the best solution to control believable agents.

2.1.5 Probabilistic Models

Probabilistic models make use of random variables and discrete or continuous probability distributions to generate output values. The distributions form the parameters of the model, defining the relation between each random variable (see figure 2.5). They have been very used, for a long time now, in hand writing (Bozinovic and Srihari, 1982; Vinciarelli, 2002; Artieres et al., 2007) and speech recognition (Levinson, 1983; Mari et al., 1996; Glass et al., 1996). Later, their use spread to robotics for the control of movement (Simmons and Koenig, 1995; Calinon and Billard, 2007) and then to the control the whole behaviour of real or virtual agents (Le Hy et al., 2004; Gorman et al., 2006a; Bauckhage et al., 2007).

In order to control the behaviour of agents, probabilistic models have to answer to the question $P(Actions|Sensors)$. It can be translated as "what is the probability of doing actions given the current value of the sensors". Note that is may be seen as an extension of a rule *IF Sensors THEN Actions*, because the probability can be translated as *IF Sensors THEN you may do Actions*. Of course, models can have internal random variables, the question being $P(Actions|Sensors, InternalState)$.

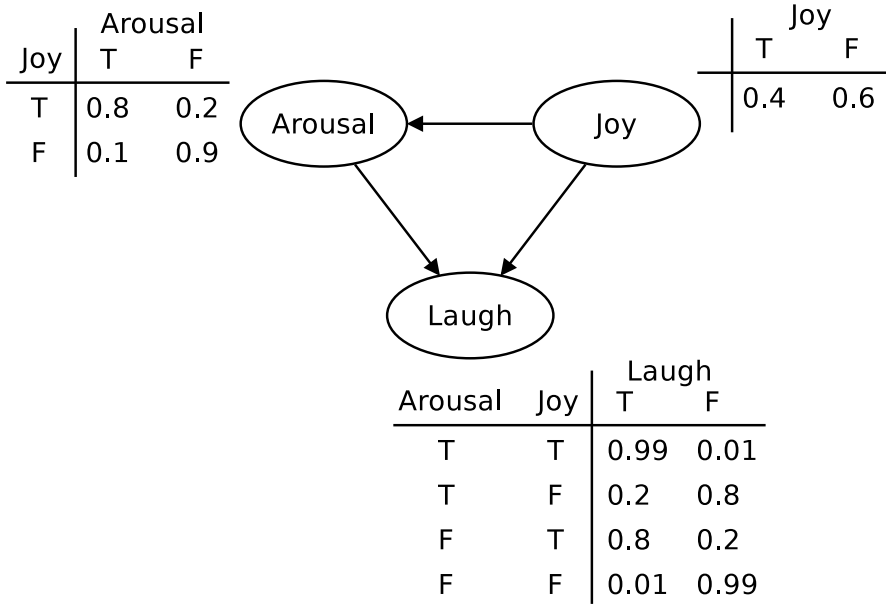


Figure 2.5: *A Bayesian network with the probability tables defining the relations between all the random variables. T and F stand respectively for true and false.*

Several probabilistic models aiming at controlling believable agents have been developed in the mid-2000s. In (Gorman et al., 2006a; Bauckhage et al., 2007), a Bayesian behaviour model previously developed in (Rao et al., 2004), is used in a video game. The model comes with a very detailed learning algorithm based on the human characteristics of imitation learning (Meltzoff and Moore, 1977). The believability of this model has been studied in (Gorman et al., 2006b), the evaluation showing that most players were fooled by the model. Another model, (Le Hy et al., 2004), is a Bayesian model which tries to keep the advantages of models from video games and combine them with the flexibility and power of Bayesian models. The model is easily understandable and parametrizable, it also offers a wide range of behaviours and comes with several learning methods. It is interesting to note that although most video games add some random picks to simulate variability and unpredictability, few or no game uses "full" probabilistic models.

The first and most obvious advantage of probabilistic models is the unpredictability of the behaviours [M1: Variability]. As each choice is based on a random pick, they are, by nature, very hard to anticipate. It is possible to define internal state to categorize behaviours in order to exaggerate them [M4: Exaggeration]. Memory can be modelled with Markovian processes [M7: Memory] where current state depend of the previous state. As probabilistic models can handle symbols and numeric values, they are flexible and distributions can be understood and modified by programmers [M3: White box]. They can model anticipation by defining goals (Rao et al., 2004) and by handling uncertain knowledge [M5: Planning]. The ability to handle uncertain knowledge makes this kind of model well adapted to learning algorithms [M2: Learning]. Last, as the models are quite modular, in the form or distributions, one can merge part of models to combine their advantages.

Their best advantage can turn out to be their worst weakness: too much randomness in the behaviour can be harmful for believability. This problem can be solved by carefully choosing or modifying the distributions for the picks to be "less random", giving more often the most likely actions. Although probabilistic models can make the agent anticipate, it is much harder to make them plan actions [M5: Planning]. However it is not certain that planning is needed for believability, short-term anticipation may be sufficient.

Probabilistic models seem the best choice for believable agents according to the result given by believability evaluation and their characteristics. Although the models are not new, their use in behaviour modelling is not widespread. As a consequence there are few models at our disposal to enhance or draw inspiration from.

	Connectionist	State transition	Production	Probabilistic
[M1: Variability]	✗	✗	✗	✓
[M2: Learning]	✓	✗	∼	✓
[M3: White box]	✗	✓	✓	✓
[M4: Exaggeration]	✗	✓	∼	✓
[M5: Planning]	✗	✗	✓	∼
[M6: Reaction time]	✓	∼	✓	✓
[M7: Memory]	∼	∼	✓	✓

Table 2.3: *Summary of the characteristics of models for the control of believable agents. The characteristics are listed at the beginning of section 2.1 and models are detailed is sections 2.1.2, 2.1.3, 2.1.4 and 2.1.5.*

Conclusion of 2.1

The summary of the characteristics of the models is shown in the table 2.3. It shows that probabilistic models are the best architecture to model the behaviour of believable agents. It is however possible to mix models from different categories to combine their strengths. For instance, connectionist models are very efficient in learning gestures or very precise part of the behaviour and then can be integrated as a part of a bigger model. It is also possible to use a production system and add some random picks to add variability and unpredictability.

The final choice of a model depends on its compatibility with a learning algorithm which fulfils the requirement for believability. Indeed, those algorithms are often designed for a type of models so the choice of a model and learning algorithm should be done together.

2.2 Algorithms to Learn Behaviours

Introduction of 2.2

According to criterion [M2: Learning], the model controlling BIBot must be able to learn to avoid mistakes and to exhibit its learning ability. A definition of learning in computer sciences is given by Mitchell (1997, page 2):

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In this thesis, T is making an agent produce believable behaviours for an avatar in a virtual environment.

Plan of 2.2

In the following, we first recall the requirements for the learning algorithm in section 2.2.1. We then discuss the possible solutions for the performance measure P in section 2.2.2 and the experience E in section 2.2.3. Finally we detail algorithms fulfilling the requirements listed in the two previous sections in section 2.2.4.

2.2.1 Requirements for the Learning of Believable Behaviours

As for models, few measures has been done to asses the believability of results produced by learning algorithm. Therefore we define requirements for learning algorithms for believable agents in table 2.4.

Learning requirements	Summary of the requirement
[L1: Believability]	Makes the model evolve into generating believable behaviours [B9: Evolution].
[L2: Believability model]	Is compatible with models fulfilling the requirement listed in section 2.1.1.
[L3: Fast]	Makes the model modify the generated behaviour for the learning to be observable by players [B10: Fast Evolution]

Table 2.4: *Requirements for the learning algorithm to make the agent behave in a believable way.*

[L1: Believability] This criteria may seem a bit vague but we must keep in mind that the algorithm must make the model generate believable behaviours. The definition, in section 1.2.1, indicate that the avatar controlled by the agent must give the illusion that it is controlled by a player. This objective

seems hardly compatible with classic learning objectives, mostly aiming at performance.

[L2: Believability model] As discussed in section 2.1, probabilistic models have good characteristics for the generation of believable behaviours. However production systems and connectionist models may also be suitable. We must look at learning algorithms for these three kind of models and find the best model/algorithm pair. Such pair should fulfil most of the requirements listed in section 2.1.1 and in this section.

[L3: Fast] The last criterion is that the learning algorithm makes the avatar change its behaviour fast enough for the learning to be believable. This means the algorithm must be computationally feasible (Angluin, 1992) and that its implementation runs fast (and not only on a super-computer).

2.2.2 Performance Measure

Now that we defined the task T and the requirements the learning algorithm must fulfil, we have to choose the performance measure P. Learning algorithms are defined as an optimization problem where the performance measure P is to be maximized. Therefore, the choice of how P is computed is critical because it influence greatly the final result. The choice of P is often very difficult because in most cases, multiple criteria have to be merged into one measure. We will study the possible measures in the following.

Optimality The most classic performance measures are based on how well the agent plays the game. The best the agent is at playing the game, the higher the score. For example for chess, if the game is won (Thrun, 1995) the score is positive and if the game is lost the score is negative. For richer environments, like video games, the performance may be more complex to measure (Stanley et al., 2005), mainly because they are many possible goals. This way of measuring the performance of the agent does not fit our goal of believability. Indeed, players do not have optimal behaviours.

Believability As we want our agent to be believable, the simplest way would be to use the measure the believability as a performance function. The major problem is that this measure is impractical for learning algorithms (Tencé and Buche, 2008). Indeed, we need either a mathematical function to use classic optimization methods or a fast and reliable measure. The only way to measure believability is by using players to assess their feelings toward the avatar they see. This measure does not give a mathematical function and it is not fast and reliable so this kind of measure cannot be used for the learning algorithm.

Mimicry As the definition of believability is to "give the illusion of being a player", it is possible to approximate it to "be like a player". Note that the two are different, the former being subjective and the latter objective. We call this measure *mimicry*: the closer the behaviour of an agent is close to a player's behaviour, the higher is the mimicry value. Although it is not our actual goal, mimicry is very close to believability and thus can be used as a performance measure by the learning algorithm, partly fulfilling [L1: Believability].

2.2.3 Learning from Experience

Now that the performance measure is defined, we have to choose from which data to learn and how to handle the data.

2.2.3.1 Which Data?

Learning from Self The agent can learn from itself by trial and error. The tries can be done in the environment or in an internal simulation. However, due to its nature, this process compute a lot of possibilities and may be slow or even infeasible with environments with lots of possibilities. It may be interesting to apply this kind of algorithm if the agent is alone in the environment.

Learning from Players A better option is for the agent to observe the other players and learn from them. As the performance measure is to look like a player, the problem is reduced to a supervised learning, each observation being a correct behaviour to learn. With this kind of data, it is possible to improve the performance of the agent in tasks T with respect to the measure P, partly fulfilling [L1: Believability].

2.2.3.2 How to Treat the Data?

Offline Learning The way the algorithm use the data is important. If we choose an *offline* learning algorithm, it will wait for all the observations to be done to produce a model having a high performance. This behaviour is not wanted because the number of observations is potentially infinite. Indeed, we do not know when all the player's behaviours are expressed and if it is even possible to express them all. As a consequence, we cannot find when to stop the observation for the algorithm to learn.

Online Learning A better solution is to make the algorithm modify the model for each observation of a player. By "observation" we mean a short example of behaviour. Too long observations would make the algorithm behave like an offline algorithm which would ruin the principle. As a consequence, a fast online algorithm should fulfill [L3: Fast].

2.2.4 Imitation Learning for Behaviour Modelling

Learning algorithm for believability should then be able to mimicry players by observing them (see section 2.2.3.1) and update the model in an online fashion (see section 2.2.3.2). According to these characteristics, the best solutions are *imitation* learning algorithms. They are based on a learning technique few animals can use (Blackmore, 1999; Meltzoff and Moore, 1977) (see figure 2.6). According to Schaal (1999); Thurau et al. (2005), imitation can also lead to believable or at least humanoid agents and robots (see figure 2.7).

Figure 2.6: *Neonates imitating different facial expressions. Taken from (Meltzoff and Moore, 1977).*

The principle of imitation learning is to record the doings and the perceived information of a player' avatar. We will call from now this player *the teacher*, even if it may not be aware that it teaches, the recording being transparent for the player. The record is given to the learning algorithm which updates accordingly the parameters of the model.

We will now present imitation algorithms fulfilling [L2: Believability model], which means that are compatible with behaviour models for believable agents. State transition and production systems do not have learning algorithms in this category. We will then present imitation learning algorithms for connectionist models (section 2.2.4.1) and probabilistic models (section 2.2.4.2).

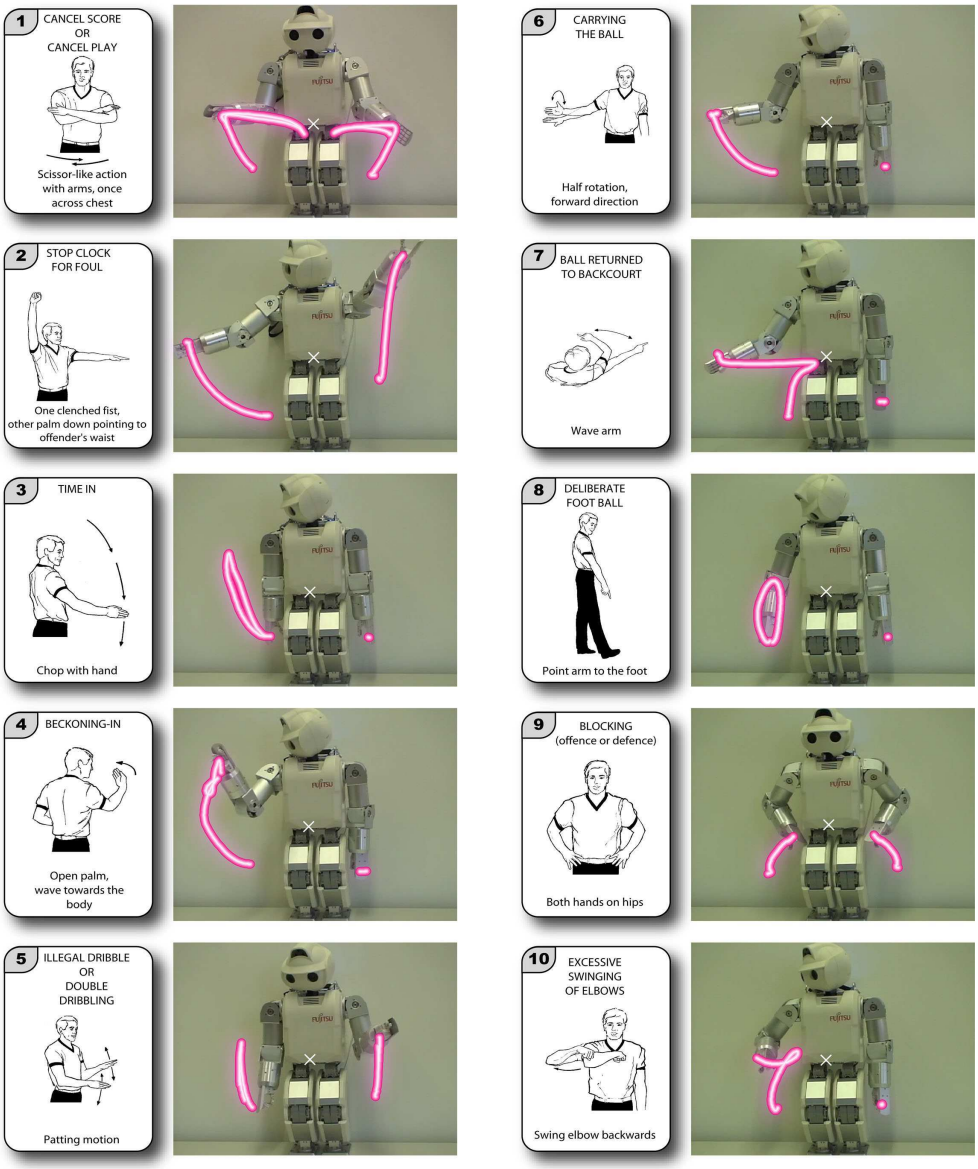


Figure 2.7: *A robot imitating 10 different gestures, after having observed 6 demonstrations for each gesture. Taken from (Calinon and Billard, 2007).*

2.2.4.1 Learning Algorithms for Connectionist Models

Learning how to Move in the Environment Movement in an environment is often both highly reactive and strategic. Thurau et al. (2004) propose an imitation learning algorithm to learn the topology of a virtual environment with a model named Growing Neural Gas (GNG). While the layout of the environment is learned by observing how players move, another algorithm learn

attraction potential for each interesting spot found by the GNG. The sum of those potentials forms a field force, attracting the agent and allowing it to find its path in the environment. The results showed that the agent is capable to find is way in the environment in a "very convincing manner" (Thurau et al., 2004).

Learning how to Act We already underlined in [B3: Variability] that the way an avatar moves is really important for the believability. Gorman and Humphrys (2007) describe how to make a neural network learn by imitation the behaviour of aiming and shooting in a First Person Shooter (FPS). The model consists of three neural networks, one for choosing a weapon, one for aiming and one to choose to fire or not. Those neural networks are trained with a Levenberg-Marquardt algorithm on players' data previously gathered and treated. The results are very encouraging as the agents are capable of short-term anticipation. Agents even copied behaviours due to the use of a mouse: right-handed players have more difficulties to follow targets travelling from the right to the left with their cursor (Gorman and Humphrys, 2007). A very similar method, giving very similar results is used in (Bauckhage and Thurau, 2004) with a mixture of experts, each expert being a neural network.

2.2.4.2 Learning Algorithms for Probabilistic Models

Learning Goal-Oriented Behaviours Being able to display understandable goals [B5: Understandable] imply being able to learn them first. Gorman et al. (2006b) modify an existing algorithm (Rao et al., 2004) for a video game. What makes it particularly interesting is that it is based on Meltzoff's work on imitation in infants (Meltzoff and Moore, 1977). The model uses two distributions, S is the state of the environment and A is the action the agent does:

$$\mathbb{P}\left(S^{t+1} \middle| S^t, A^t\right) \tag{2.1}$$

$$\mathbb{P}\left(A^t \middle| S^t, S^G\right) \tag{2.2}$$

The distribution (2.1) gives the consequence of an action on the environment and the distribution (2.2) gives the action to do to satisfy the goal state S^G. With this second distribution it is possible for the agent to guess the teacher's goals to better understand and learn his/her behaviour. The model after learning is able to fool other players into thinking the agent is human Bauckhage et al. (2007). It seems, however, to have problem to generalize, reproducing the learned behaviours only in the same context.

Learning Temporal Behaviours As explained in [B8: Memory], the agent must exhibit a consistent behaviour over time. The learning algorithm must be able to handle behaviours which last over time. Le Hy et al. (2004) propose

a model close to a Hidden Markov Model (HMM) (see section 2.3.1.1) and a learning algorithm which is based on (Florez-Larrahondo, 2005). The learning algorithm allows the model to learn the sequencing of the decisions taken by the teacher. The believability of the results has not been clearly assessed, but it seems to be a good way to learn convincing behaviours.

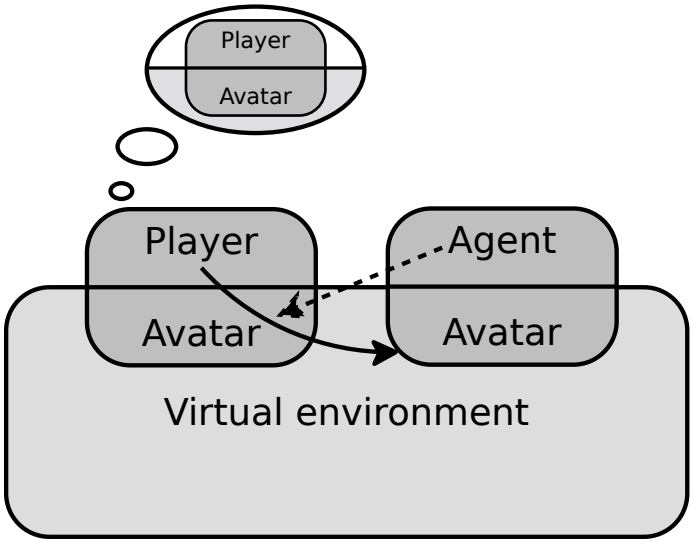


Figure 2.8: *Interest of the imitation learning considering the goal of believability. The plain arrow represents the player observing the avatar of the agent in the virtual environment. The dashed arrow represents the agent observing and imitating the player's avatar in order to appear believable.*

Conclusion of 2.2

According to the study of possible solutions, probabilistic models and their imitation learning algorithms seems to be able to model and learn the whole teacher's behaviour in order to produce believable behaviour (see figure 2.8). Connectionist models should not be forgotten because they can be used to learn specific part of the behaviour in a very efficient way.

Among the probabilistic solutions, Le Hy et al. (2004)'s proposal seems the most suited to our needs. It can learn by imitation the teacher's behaviour and handle the different sequences of behaviours. The two presented connectionist models and learning algorithm (Thurau et al., 2004; Gorman and Humphrys, 2007) could be merged with Le Hy's work to improve the results.

2.3 Le Hy's Work

Introduction of 2.3

The model and the learning algorithm proposed by Le Hy et al. (2004) fulfil most of the requirements we defined in section 2.1.1 and 2.2.1. As a consequence, Le Hy's work is a good candidate to comply to the requirements in section 1.2, generating believable behaviours. Therefore this study will serve as a base for further developments in this thesis. We will point out the weaknesses of the model and the learning algorithm and try to find solution to fix them. Before that we will detail Le Hy's work.

Plan of 2.3

In this section we present the principle of the model and its underlying theory, the Input-Output Hidden Markov Model (IOHMM), in section 2.3.1. Then, we detail the two associated imitation learning algorithms in section 2.3.2: Laplace's rule of succession and the Incremental Baum-Welch algorithm (IBW).

2.3.1 Principle of the Model

In order to better understand Le Hy's model, we present the theoretical basis behind the model. It will allow to introduce the different notations and the underlying concepts.

2.3.1.1 Hidden Markov Models (HMMs)

Hidden Markov Models (HMMs) (Rabiner, 1989) are well known probabilistic models which are used in a wide range of domains: speech recognition (Rabiner, 1989), prediction in biology (Krogh et al., 2001), signal processing (Crouse et al., 1998), analysis of sequence of images (Yamato et al., 1992), etc. They are not used for behaviour modelling because, as we will see, they cannot handle inputs. They model dynamic systems which internal state is unknown but this state influence the emission of observable events. A HMM defines a set of states $d_1, d_2, ..., d_{|\mathcal{D}|}$ and a set of observation $a_1, a_2, ..., a_{N_A}$. At each time step t, the random variable D^t gives the value of the state and the variable A^t gives the value of the observation (see figures 2.9, 2.10, 2.11 and 2.12).

The hypothesis are:

- The choice of D^t depends only on D^{t-1}, according to the distributions $m_{ij}^t = \mathbb{P}\left(D^t = d_i \middle| D^{t-1} = d_j\right)$. The letter m stands for *Markovian*.
- The choice of A^t depends only on D^t, according to the distributions $o_i^t(a) = \mathbb{P}\left(A^t = a \middle| D^t = d_i\right)$. The letter o stands for *output*.

HMMs can be expressed as graphs with all the values (see figure 2.9) or using the Dynamic Bayesian Network (DBN) notation (Murphy, 2002, page 20) (see figure 2.11).

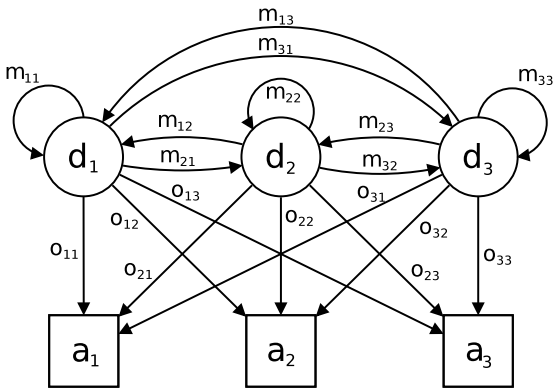


Figure 2.9: *A HMM represented in a classic way with 3 hidden states and 3 observable events.*

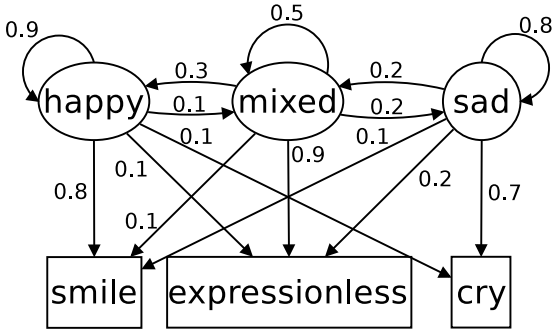


Figure 2.10: *A concrete example of a HMM with 3 hidden states representing feelings and 3 visible observations representing the expressions. Edges that are not drawn express a probability of 0.*

2.3.1.2 Input-Output Hidden Markov Models (IOHMMs)

As Hidden Markov Models (HMMs) are only designed to generate outputs, there are not suited for dynamic systems which also take into account external data, like for example decision taking systems. An extension of HMMs allows the model to use the value of inputs for the generation of outputs: the Input-Output Hidden Markov Models (IOHMMs) (Bengio and Frasconi, 1995). An IOHMM defines, on top of the usual definitions for a HMM, a set of inputs $s_1, s_2, ...s_N$. A new random variable S^t gives the value of the input at time t.

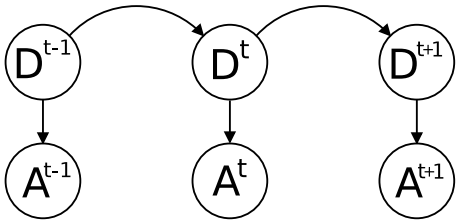


Figure 2.11: *A HMM represented as a DBN (Murphy, 2002, page 20). Neither the number of hidden states and observations nor the probabilities are specified.*

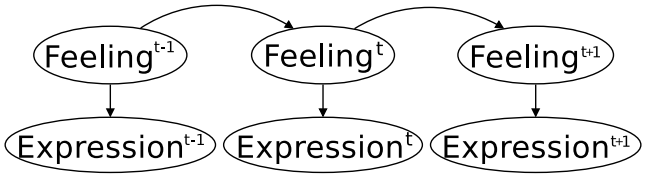


Figure 2.12: *A concrete example of a HMM represented as a DBN (Murphy, 2002, page 20). Neither the number of values for hidden feelings and observable expressions nor the probabilities are specified.*

The hypothesis are:

- The choice of D^t depends only of D^{t-1} and S^t, according to the distribution $m^t_{ij}(s) = \mathbb{P}\left(D^t = d_i \,\middle|\, D^{t-1} = d_j, S^t = s\right)$.
- The choice of A^t depends only of D^t and S^t, according to the distribution $o^t_i(a,s) = \mathbb{P}\left(A^t = a \,\middle|\, D^t = d_i, S^t = s\right)$. Sometimes A^t depends only of D^t so we have $o^t_i(a,s) = \mathbb{P}\left(A^t = a \,\middle|\, D^t = d_i\right)$.

As the structure of the model becomes more and more complex, it is preferable to represent the model only with a the DBN notation (Murphy, 2002, pages 25-26) (see figure 2.13 and 2.14). The parameters of the models should be specified elsewhere.

2.3.1.3 Le Hy's Model

Le Hy's Model is very similar to an IOHMM, the only different is that Le Hy's model have multiple inputs and outputs. The model defines the random variables $S^t_0, .., S^t_{N_S}$, each one taking the values of a sensor at time t. Sensors give information on internal and environment's state, like for instance which object the avatar has in its hand or the position of an other avatar. Similarly, the random variables $A^t_1, ..., A^t_{N_A}$ are defined, each one taking the value of an action at time t. Actions can be to rotate by a certain amount of degrees, to

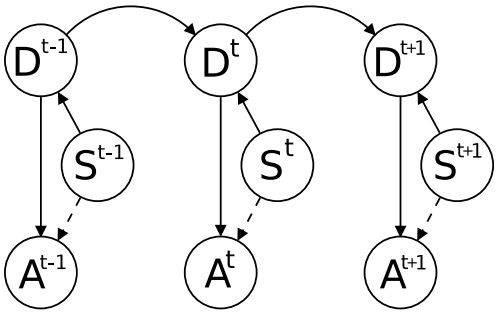


Figure 2.13: *An IOHMM represented as a DBN (Murphy, 2002, pages 25-26). The number of hidden states, inputs and outputs is not specified. The dashed line means A and S may be independent given D depending on the system to be modelled.*

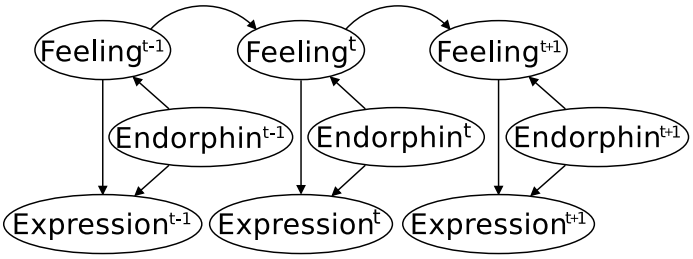


Figure 2.14: *An example of an IOHMM represented as a DBN (Murphy, 2002, pages 25-26) which models the expression of a humanoid given its level of endorphins.*

jump or to change the object carried in the avatar's hand. Both sensors and actions random variables take discrete values.

In order to make the agent's behaviour more complex, the notion of decision has been introduced, the associated hidden random variable is named D^t and may take different values like searching for an object or fleeing. It is this random variable which will allow the model to clearly define behaviours which will make the agent's intention more understandable to observers.

As in IOHMMs, the value of the current decision (hidden variable), D^t, is chosen according to the value of the sensors (inputs) and the previous decision following the probability distribution $\mathbb{P}\left(D^t \left| S_0^t, .., S_{N_S}^t, D^{t-1}\right.\right)$. In order to reduce the number of parameters, this distribution is first expressed as follows:

$$\mathbb{P}\left(D^t \left| S_0^t, .., S_{N_S}^t, D^{t-1}\right.\right) \tag{2.3}$$

$$= \frac{\mathbb{P}(D^t, S_0^t, .., S_{N_S}^t, D^{t-1})}{\mathbb{P}(S_0^t, .., S_{N_S}^t, D^{t-1})} \tag{2.4}$$

$$\propto \mathbb{P}\left(S_0^t, .., S_{N_S}^t \left| D^t \right.\right) \mathbb{P}\left(D^t \left| D^{t-1} \right.\right) \tag{2.5}$$

The symbol $\propto$ means "proportional to".

As there may be a lot of sensory random variables, $\mathbb{P}\left(S_0^t, .., S_{N_S}^t \left| D^t \right.\right)$ may take too many values to be tractable. Le Hy uses the notion of Inverse Programming (IP) (Le Hy et al., 2004) to reduce the complexity: each $S_i^t, i \in [\![1, N_S]\!]$ is assumed to be independent given D^t:

$$\mathbb{P}\left(S_0^t, .., S_{N_S}^t \left| D^t \right.\right) = \prod_{i=0}^{N_S} \mathbb{P}\left(S_i^t \left| D^t \right.\right) \tag{2.6}$$

then, we have:

$$\mathbb{P}\left(D^t \left| S_0^t, .., S_{N_S}^t, D^{t-1} \right.\right) \propto \mathbb{P}\left(D^t \left| D^{t-1} \right.\right) \prod_{i=0}^{N_S} \mathbb{P}\left(S_i^t \left| D^t \right.\right) \tag{2.7}$$

In order to obtain the exact value of the distribution, we simply have to marginalize over D^t because we have the constraint:

$$\sum_d \mathbb{P}\left(D^t = d \left| S_0^t, .., S_{N_S}^t, D^{t-1} \right.\right) = 1 \tag{2.8}$$

Once the value of D^t is chosen, the model then decides which actions should be carried out. The value of each action is chosen following the distribution $\mathbb{P}\left(A_i^t \left| S_0^t, .., S_n^t, D^t \right.\right)$. Again, to reduce the complexity, Le Hy uses the notion of Fusion by Enhanced Coherence (FEC) (Le Hy, 2007, page 55, in French). The influence of each sensory information is separated following the formula (see figure 2.15):

$$\mathbb{P}\left(A_i^t \left| S_0^t, .., S_{N_S}^t, D^t \right.\right) \propto \prod_j \mathbb{P}\left(A_i^t \left| S_j^t, D^t \right.\right) \tag{2.9}$$

Again, a normalization over A_i^t gives the exact value.

To sum up, the model, which can be categorized as an IOHMM, uses a very simple algorithm (see figure 2.16). The relation between the random variables are summarized in figure 2.17 and an example is given in figure 2.18. The whole model is composed of three types of parameters which are probability distributions:

- $\mathbb{P}\left(D^t \left| D^{t-1} \right.\right)$

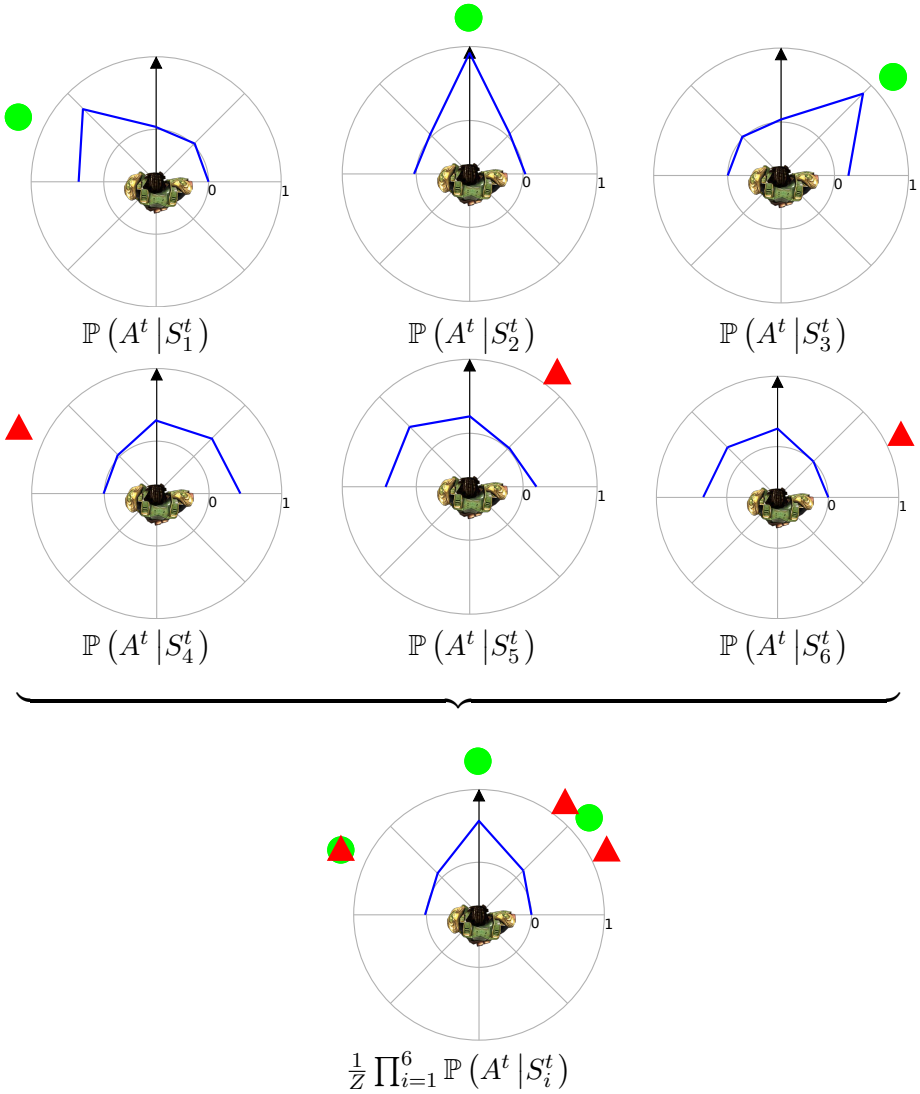


Figure 2.15: *Principle of the FEC (Le Hy, 2007, page 55, in French). Each diagram represents the probability (blue line) for 5 rotation commands, the avatar viewed from the top: turn left 90°, turn left 45°, do not turn, turn right 45° and turn right 90°. The black arrow represents the avatar's facing direction. Green circles are attractor and red triangles repulsors. Each command is specified for a single sensor's value (the 6 diagrams at the top), the resulting FEC is the bottom diagram.*

- $\mathbb{P}\left(S_i^t \left| D^t \right.\right)$
- $\mathbb{P}\left(A_i^t \left| S_j^t, D^t \right.\right)$

while agent in world **do**
  s ← agent's sensors
  Pick d using (2.7)
  for all $i \in [\![1, N_A]\!]$ **do**
    Pick a_i using (2.9)
    Make avatar do (a_i)
  end for
  $d^{t-1} \leftarrow d$
end while

Figure 2.16: *Algorithm of the Le Hy's model. It is possible to choose the way the value are picked: randomly following the distribution, using the maximum value in the distribution, etc.*

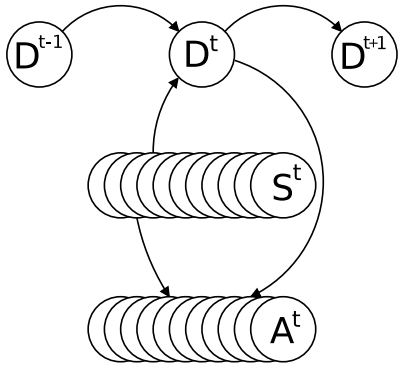


Figure 2.17: *Summary of the influences between Le Hy's model's variables (Le Hy et al., 2004). S is for sensors, A for actions and D for decisions. The model is represented as a DBN for the sake of simplicity.*

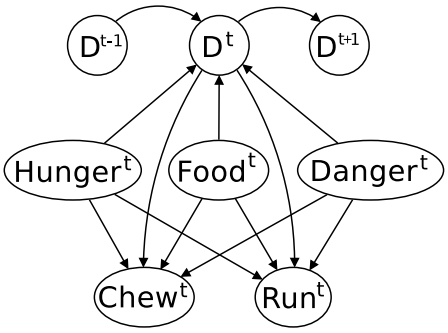


Figure 2.18: *Example of a use of Le Hy's model. The values for D can be* Flee *and* Eat.

2.3.2 Imitation Learning Algorithms

The three distributions $\mathbb{P}\left(D^t \left| D^{t-1}\right.\right)$, $\mathbb{P}\left(S_i^t \left| D^t\right.\right)$ and $\mathbb{P}\left(A_i^t \left| S_j^t, D^t\right.\right)$ can be specified manually but Le Hy developed also two learning algorithms to learn them. According to Le Hy, results seems to be better in term performance with learned parameters (Le Hy, 2007, page 103, in French). By monitoring at each time step the values for S and A for a teacher's avatar, it is possible to update the value of the parameters. The learning algorithms developed by Le Hy are based on Laplace's rule of succession for the learning of $\mathbb{P}\left(A_i^t \left| S_j^t, D^t\right.\right)$ in section 2.3.2.1 and the IBW (Florez-Larrahondo, 2005) for the learning of $\mathbb{P}\left(D^t \left| D^{t-1}\right.\right)$ and $\mathbb{P}\left(S_i^t \left| D^t\right.\right)$ in section 2.3.2.3.

2.3.2.1 Rule of Succession

For the learning of the distributions for the choice of actions, $\mathbb{P}\left(A_i^t \left| S_j^t, D^t\right.\right)$, Le Hy uses an algorithm based on Laplace's rule of succession. Giving a random variable X which can take the values $success$ or $failure$, this rule tries to estimate the next most likely result after n picks with s success. The formula is:

$$P(X = success) = \frac{s+1}{n+2} \tag{2.10}$$

Le Hy uses a generalized version of Laplace's rule of succession. Instead of working with a binary random variable, it works with multiple random variables which can have any number of possibility. The estimation of the distribution is done with the formula:

$$\mathbb{P}\left(A_i^t = a \left| S_j^t = s, D^t = d\right.\right) = \frac{1 + n(a,s,d)}{|\mathcal{A}_i^t| + n(s,d)} \tag{2.11}$$

Where $|\mathcal{A}_i^t|$ is the number of values the random variable can take, $n(a,s,d)$ is the number of time the triple a, s and d is observed and $n(s,d)$ the number of time the pair s and d is observed.

It is not clear how Le Hy's algorithm find the value of D^t, as it is a hidden variable. The most likely solution is that he uses a decision detection heuristic (Le Hy, 2007, pages 82-83, in French). It is also possible to retrieve the probability of D^t from the algorithm that will described in 2.3.2.3, but that would mean the two learning algorithms would be interlaced. This could be problematic because each learning algorithm would use distributions which the other algorithm is learning. The convergence and the validity of such a learning could not be assured.

2.3.2.2 Baum-Welch algorithm (BW) and Incremental Baum-Welch algorithm (IBW)

For the learning of the distributions used for the choice of the decisions, $\mathbb{P}\left(D^t \left| D^{t-1}\right.\right)$ and $\mathbb{P}\left(S_i^t \left| D^t\right.\right)$, Le Hy uses an algorithm based on the BW (Baum et al., 1970). This algorithm is designed to learn the parameters of a HMM. This algorithm can be categorized as an Expectation-Maximization algorithm (EM) (more detail in section 3.4). It iterates, making the parameters of the HMM converge toward a local optimum. In order to do so, the BW alternates between two steps:

- Computes estimators using the parameters
- Updates the parameters using the estimators

We recall that the distributions of a HMM are :

- The Markovian distribution: $m_{ij}^t = \mathbb{P}\left(D^t = d_i \left| D^{t-1} = d_j\right.\right)$
- The output distribution: $o_i^t(a) = \mathbb{P}\left(A^t = a \left| D^t = d_i\right.\right)$

The estimators are based on the forward-backward algorithm (Rabiner, 1989). Let a sequence of observations $\mathcal{A}^{1,T} = a^1, a^2, ...a^T$ of length T. The parameters of the model at iteration n are named Φ^n. The forward variable is defined as:

$$\alpha_i^t = \mathbb{P}\left(\mathcal{A}^{1,T}, D^t = d_i \left| \Phi^n\right.\right) \tag{2.12}$$

And the backward variable as:

$$\beta_i^t = \mathbb{P}\left(\mathcal{A}^{1,T} \left| D^t = d_i, \Phi^n\right.\right) \tag{2.13}$$

α_i^t is the probability for the model to produce the observed outputs from the beginning of the recorded sequence to t and to be in a specific hidden state at t. β_i^t is the probability for the model to produce the observed outputs from $t+1$ to the end of the recorded sequence given the model to be in a specific hidden state at t.

It is then possible to estimate the probability that the model is in the state d_i at t given the observations and the actual parameters of the model:

$$\gamma_i^t = \mathbb{P}\left(D^t = d_i \left| \mathcal{A}^{1,T}, \Phi^n\right.\right) \tag{2.14}$$

$$\propto \alpha_i^t \beta_i^t \tag{2.15}$$

It is also possible to compute the probability that the model makes the transition between the state d_j at time $t-1$ and state d_i at time t:

$$\xi_{ij}^t = \mathbb{P}\left(D^t = d_i, D^{t-1} = d_j \left| \mathcal{A}^{1,T}, \Phi^n\right.\right) \tag{2.16}$$

$$\propto \alpha_j^{t-1} m_{ij} o_i(c^t) \beta_i^t \tag{2.17}$$

Now that the estimators are defined, it is possible to update the parameters for the model to be more likely to produce the observed sequence:

$$m'_{ij} = \frac{\sum_{t=2}^{T} \xi_{ij}^t}{\sum_{t=1}^{T-1} \gamma_j^t} \tag{2.18}$$

and:

$$o'_i(a) = \frac{\sum_{t=1}^{T} \gamma_i^t \mathbb{1}_a(a^t)}{\sum_{t=1}^{T} \gamma_i^t} \tag{2.19}$$

Where $\mathbb{1}_x(y)$ is the indicator function which takes the value 1 if $x \in y$, or by abusing the notation if $x = y$, and 0 if not.

The Incremental Baum-Welch algorithm (IBW) is an online version of the BW: it updates the parameters of the model for each given observation of the value of A^t, instead of waiting for a whole sequence $\mathcal{A}^{1..T}$. The IBW converges faster and requires less computation power. Of course, it comes at a cost: the expectation step uses simplified estimators (see equation 2.21). Indeed, at time t, β_i^t cannot be computed because the algorithm does not know what will happen after t. The algorithm also makes some simplification during the update in order to modify the parameters in an iterative way (see equation 2.30).

2.3.2.3 Le Hy's Learning Algorithm

Le Hy uses a modified version of the IBW to learn the distributions related to the decisions. Indeed, the algorithm must take into account the value of multiple actions $\mathcal{A}^{1..T} = \{a_i^1, a_i^2, ..., a_i^T\}, i \in [\![1, N_A]\!]$ and multiple sensors $\mathcal{S}^{1..T} = \{s_i^1, s_i^2, ..., s_i^T\}, i \in [\![1, N_S]\!]$. Note that now the current time step is considered to be T, as the algorithm do not have information about the future. The forward variable is defined in a BW fashion:

$$\alpha_i^t = \mathbb{P}\left(\mathcal{A}^{1..T}, D^t = d_i \,\middle|\, \mathcal{S}^{1..T}, \Phi^n\right) \tag{2.20}$$

As in (Florez-Larrahondo, 2005, page 55), the following hypothesis is made:

$$\forall t,\ \beta_i^t = 1 \tag{2.21}$$

It is then possible to estimate the probability that the model is in the state d_i at t given the observations and the actual parameters of the model:

$$\gamma_i^t = \mathbb{P}\left(D^t = d_i \,\middle|\, \mathcal{A}^{1..T}, \mathcal{S}^{1..T}, \Phi^n\right) \tag{2.22}$$

$$\propto \alpha_i^t \tag{2.23}$$

It is also possible to compute the probability that the model makes the transition between the state d_j at time $t-1$ and state d_i at time t:

$$\xi_{ij}^t = \mathbb{P}\left(D^t = d_i, D^{t-1} = d_j \,\middle|\, \mathcal{A}^{1..T}, \mathcal{S}^{1..T}, \Phi^n\right) \tag{2.24}$$

$$\propto \alpha_j^{t-1} m_{ij}(s^t) o_i(a^t, s^t) \tag{2.25}$$

Like the BW it is possible to update the parameters of the model with the estimators. We note $\mathbb{P}_n(A|B)$ the distribution as the n^{th} iteration of the learning algorithm:

$$\mathbb{P}_{n+1}\left(D^t = d_i \middle| D^{t-1} = d_j\right) \tag{2.26}$$

$$= \frac{\sum_{t'=2}^{T} \xi_{ij}^{t'}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} \tag{2.27}$$

$$= \frac{\sum_{t'=2}^{T-1} \xi_{ij}^{t'}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} + \frac{\xi_{ij}^{T}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} \tag{2.28}$$

$$= \frac{\sum_{t'=2}^{T-1} \xi_{ij}^{t'}}{\sum_{t'=1}^{T-2} \gamma_j^{t'}} \frac{\sum_{t'=1}^{T-2} \gamma_j^{t'}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} + \frac{\xi_{ij}^{T}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} \tag{2.29}$$

which is assumed to be equal to (Florez-Larrahondo, 2005, page 53):

$$= \mathbb{P}_n\left(D^t = d \middle| D^{t-1} = d'\right) \frac{\sum_{t'=1}^{T-2} \gamma_j^{t'}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} + \frac{\xi_{ij}^{T}}{\sum_{t'=1}^{T-1} \gamma_j^{t'}} \tag{2.30}$$

$$\propto \mathbb{P}_n\left(D^t = d \middle| D^{t-1} = d'\right) + \frac{\xi_{ij}^{T}}{\sum_{t'=1}^{T-2} \gamma_j^{t'}} \tag{2.31}$$

Similarly:

$$\mathbb{P}_{n+1}\left(S_i^t = s \middle| D^t = d\right) \tag{2.32}$$

$$\propto \mathbb{P}_n\left(S_i^t = s \middle| D^t = d\right) + \frac{\gamma_j^T \mathbb{1}_s(s_i^T)}{\sum_{t'=1}^{T-2} \gamma_j^{t'}} \tag{2.33}$$

The algorithm is thus able to update the parameters of the model for each observation $\{\mathcal{S}^t, \mathcal{A}^t\}$. As all the variables and estimator can be computed iteratively, the algorithm has only a few computation to make to update the parameters.

These advantages mainly comes from the hypothesis $\beta(t) = 1$. In our case $\beta(t)$ gives the chances of taking a certain decision at t knowing what the demonstrator did at $t+1...T$. This estimator should not be approximated so roughly: for instance, if the teacher is looking for something specific, the learning algorithm cannot know what it is until it is picked up. Changes should be made on the algorithm to avoid such simplistic hypothesis.

Conclusion of 2.3

Le Hy's model rely on well-known models (HMM, IOHMM) whose theory has been studied for a long time. This makes the foundation of the model trustworthy. The architecture of the model is quite simple, making it easy to understand, parametrize and modify [M3: White box]. This simplicity may however also a weakness if the generated behaviours are not complex enough to meet all the believability requirements.

The learning is also based on widely-used algorithms (Laplace's rule of convergence, BW) whose mathematical properties have been studied and demonstrated. The algorithms are fast and are able to learn all the parameters of the model. However, the two algorithms could gain in efficiency if merged together. Moreover, in order to learn more complex behaviours, some hypothesis introduced in the IBW should be dropped.

2.4 Believability of Agents Using Le Hy's Model

Introduction of 2.4

Now that we described the details of both the model and the learning algorithm, we will now try assess the believability of the behaviour generated by the model. The only way to evaluate the model is to implement it and to observe the behaviour it produces. Implementing the model also allows us to evaluate the practicality and flexibility of the model.

Plan of 2.4

In this section we describe the detail of our implementation of the model in section 2.4.1, underlining the flexibility the model allows. We then point out the weakness of the model in term of behaviour and in the implication on the implementation in section 2.4.2.

In order to implement Le Hy's model, we used the game Unreal Tournament 2004 (UT2004), a newer version of the game Le Hy used for his model (see figure 2.19). We choose this game because it fits our needs (see section 1.1.2) and because a set of tools, Pogamut (Burkert et al., 2007), have been developed for researchers to use the game as a testbed for agents.

2.4.1 BIBot: an Implementation of Le Hy's Model

The model was built using the Pogamut framework. This framework was developed in order to interact easily with the Gamebots interface that controls remote agents in UT2004. The communication between the model and the environment is realized by messages over the network (see figure 2.20).

Figure 2.19: *The view a of player inside the game Unreal Tournament 2004.*

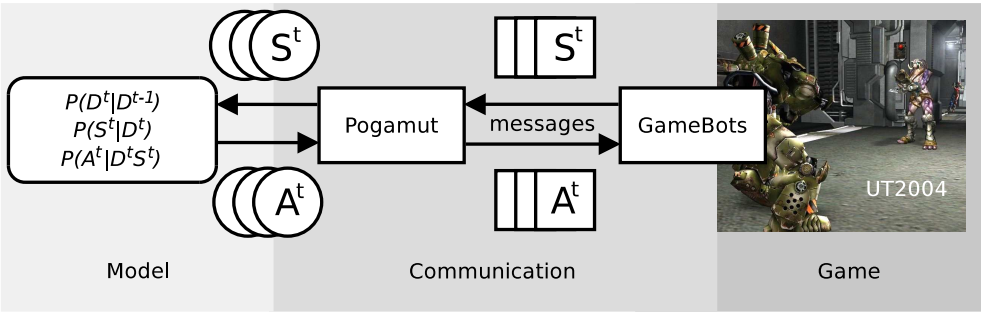


Figure 2.20: *The communication between the model and the video game is done via Gamebots and Pogamut.*

The architecture is quite simple, we designed BIBot which lays on the definition of an agent provided by Pogamut (see figure 2.21). That gives us access to the key components of an UT2004 avatar which are the body and the memory. On one side, the body deals with all the actions and on the other side the memory store all the perceptions the avatar can get during its evolution in the virtual environment.

In our model, called BIBot for Bayesian Inference based Bot, we decided to categorize these aspects in DECISIONS and SENSORS as suggested in (Le Hy et al., 2004). As the actions are already implemented in Pogamut, we did not have to create the associated classes, that is the reason they do not appear in the architecture.

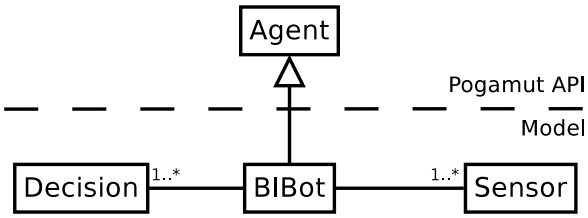


Figure 2.21: *BIBot inherits from Pogamut agent.*

We insisted on making it modular so the model can be extended by the addition of new decisions or sensors (see figure 2.22 and figure 2.23).

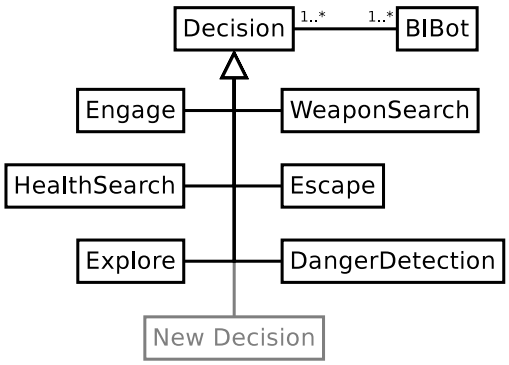


Figure 2.22: *New decisions can easily be integrated by extending the* Decision *class.*

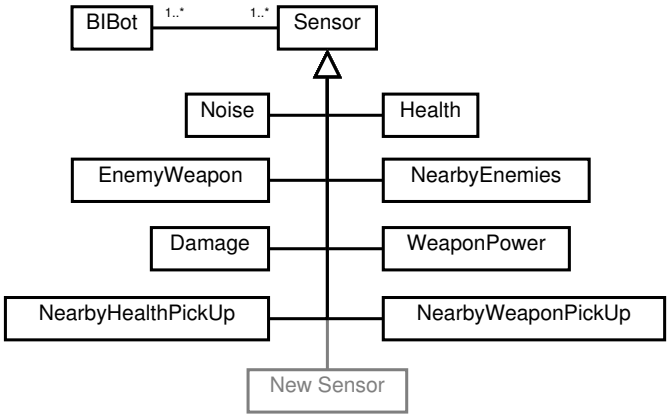


Figure 2.23: *New sensors can easily be integrated by extending the* Sensor *class.*

The probability distribution are implemented as arrays which are in the BIBot class. When adding a decision, a sensor or an action, the corresponding probability arrays must be added leaving the other parameters mostly untouched. The model and its implementation are then easily modifiable, allowing people with few programming knowledge to modify the behaviour.

In order to control the decision sequencing we made a small user interface (see figure 2.24), that gives feedback on previous decision, current running decision and the different sensors values we want to show such as the health level, the noise sensor or if our bot is being hit. We also used an external view of the agent from the game (figure 2.25) to be able to observe the behaviour of BIBot.

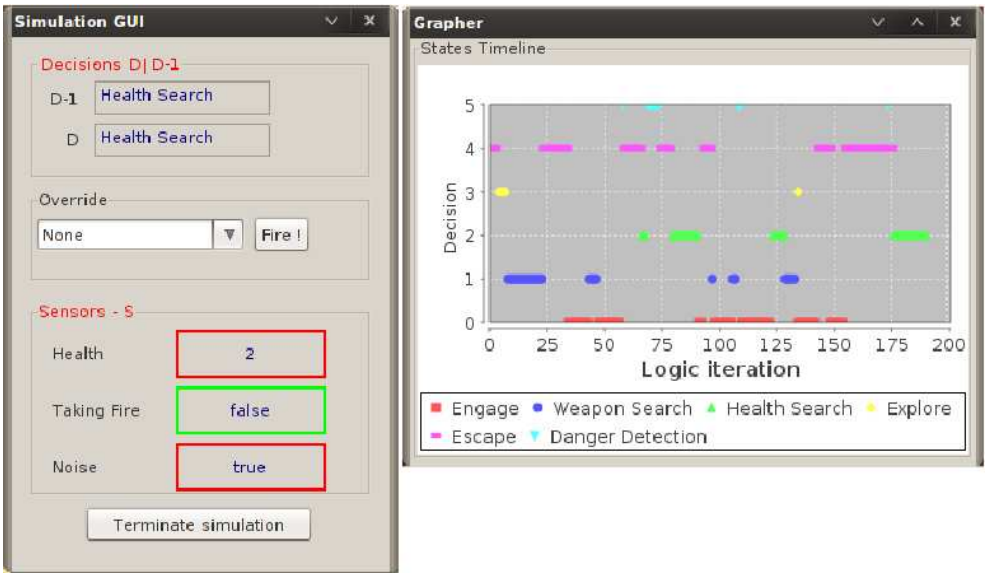


Figure 2.24: *Overview of the monitoring interface. At the left we have the current value of decisions and some sensors. At the right we have a summary of the decisions taken over time.*

For this implementation, we used the probabilities tables values as specified in (Le Hy, 2007, pages 97–102, in French) (aggressive manual specification). When BIBot is evolving alone in the environment hence does not feel in danger, the decisions are switching between *Weapon Search* and *Explore* as we can see on figure 2.26. That balance can be disturbed by adding in the environment another player (human or not) which will be considered by BIBot as a threat. Therefore its behaviour changes according to the modification of the surrounding environment (see figure 2.27).

Figure 2.25: *View from the game UT2004 with an external view of the agent and debugging information from Pogamut. This kind of view allow the observation of the agent without perturbing its environment.*

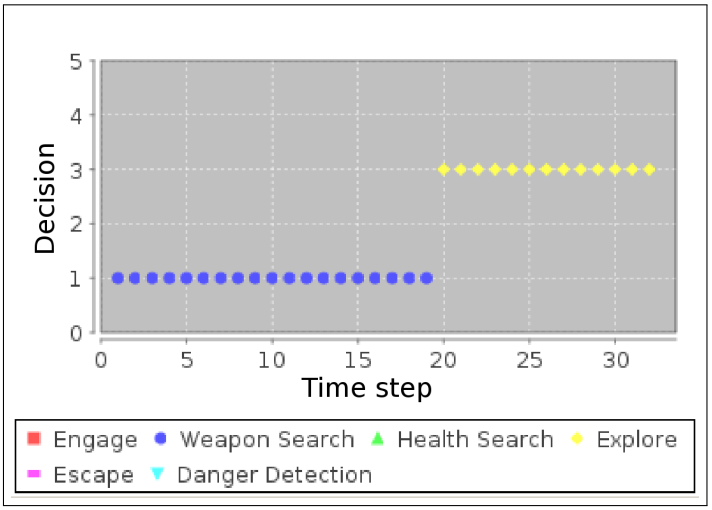


Figure 2.26: *Decision sequencing before perturbation. The perturbation is the view of a opponent's avatar. The agent is switching between different decisions.*

Figure 2.27: *Decision sequencing after perturbation. The perturbation is the view of a opponent's avatar at time step 85.*

2.4.2 Evaluation and Limits of the Model

BIBot allowed us to spot the strengths and weaknesses of both the architecture and the resulting behaviours of Le Hy's model. As the architecture of the model reminds of a FSM, it is easy to understand and to adjust the parameters. The model is modular, allowing the programmer to add or remove sensors, decisions and actions without modifying the code too much.

However, the behaviours produced by the model in the game can easily be spotted as artificial by casual and regular players. That is the first of the two main reasons we preferred not to run a complete experiment of the believability of the model. The second reason is that such experiment is very complex and time-consuming (Tencé et al., 2010). However, in the future, we will assess the believability of both Le Hy's model and our model to confirm our first impressions.

[P1: Navigation] The paths the agent uses to go from one point of the environment to another do not look like the ones a player would take. This problem does not comes from the model itself but from the representation it uses for the environment. Indeed, the agent uses navigation points placed by the designers of the environment which may not represent well how players prefer to use the environment.

[P2: Sensors] Even if it is easy to add sensors to the model, increasing the number of perceptions makes the model more and more complex. We found that sensors were useful for the choice of the decisions but not for the choice

Problems	Summary of the problem
[P1: Navigation]	The agent has problem navigating in environments.
[P2: Sensors]	The sensors are not well organized.
[P3: Expressiveness]	The FEC does not provide enough expressiveness for the agent to be believable.
[P4: Scaling]	The IP have problem handling many sensors.
[P5: Readability]	The IP is not easy to read for novices.
[P6: Learning]	The learning algorithm use strong hypothesis which may hinder its capabilities.

Table 2.5: *List of the noticed limitations with Le Hy's model.*

of actions and *vice-versa*. This is confirmed by the fact Le Hy uses in his implementation different values for the random variables S_i depending if they are used for the choice of decision (Le Hy, 2007, pages 60–70, in French) or actions (Le Hy, 2007, pages 36–54, in French). This should be clearly defined in the model.

[P3: Expressiveness] The FEC does not provides enough expressiveness: each sensory data is considered to have the same importance. For example, if there are two attractors, one straight ahead and the other behind, the agent may constantly switch between the two (see figure 2.28). A real player will choose to focus on the attractor ahead, supposing the attractors are of the same strength.

[P4: Scaling] For the IP to be valid, all the sensors S_i^t must be conditionally independent given the decision D^t. This hypothesis is very strong but can work for few sensors. However, the more the sensors, the higher the chances the hypothesis is wrong, which can make the model produce unbelievable behaviours.

[P5: Readability] The IP technique makes the parameters not very understandable [M3: White box]. It is not natural to ask "what should I see if I am taking this decision?", which correspond to the parameter $\mathbb{P}\left(S_i^t \middle| D^t\right)$. A much more natural way it to express $\mathbb{P}\left(D^t \middle| S_i^t\right)$ which corresponds to "what should I decide to do if I see that?".

[P6: Learning] In order to make the behaviour more complex and believable, one must add sensors, actions and decisions. However, the number of parameters rapidly becomes intractable for a programmer to specify them manually. The learning algorithms Le Hy developed are very simple and the parameters could be learned much more precisely given the observations of a

teacher's avatar [B9: Evolution]. A lot of hypothesis are used to simplify the computation which may be harmful for the learning. Also, all the distributions may be learned with one algorithm instead of splitting the learning in two different algorithms: the interfacing between the algorithms would not be needed any more and the convergence could be proven more easily.

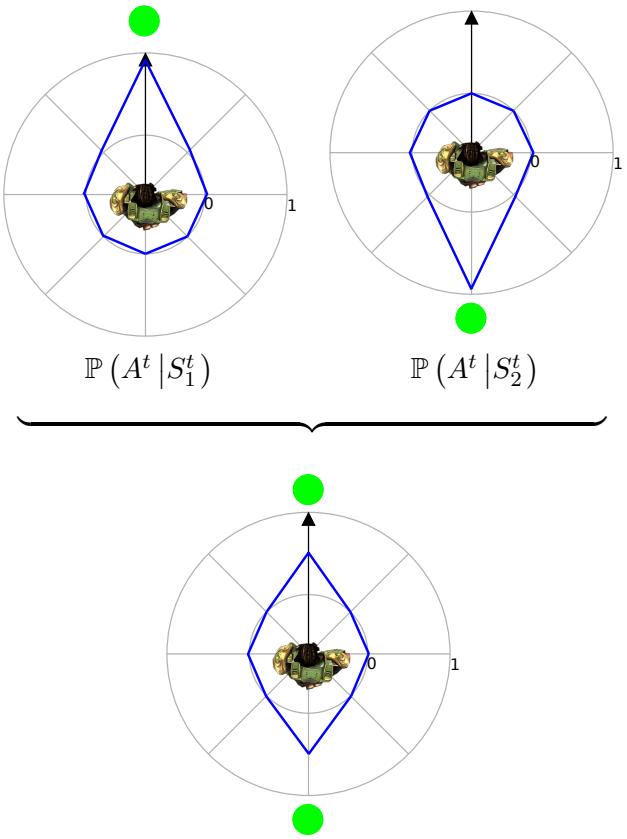


Figure 2.28: *An illustration of the problem with FEC: on the top, each distribution (blue line) gives a believable direction to go for a single attractor (green dot). On the bottom the FEC does not give a believable action because the agent may constantly switch between the two attractors, oscillating constantly.*

Conclusion of 2.4

Le Hy's approach allows an easy implementation and a great flexibility. Sensors and actions can be added without rethinking all the architecture, by adding only a class and few parameters. The model is generic and can be adapted to different virtual environments.

The behaviours generated by the model are however too simple to sustain the illusion of believability very long. This is due to the inner mechanisms of the model, the FEC and the IP. The parameters of the model could also be easier to read for non-programmers to be able to modify the behaviour. The model could be refined to express more complex behaviour without adding too much parameters. Finally, the learning could be done by one algorithm making the learning more efficient.

Conclusion of 2

In this chapter we saw that probabilistic models are a very good solution for the learning and the generation of believable behaviour. Connectionist model can also help in developing part of the model due to their very good ability to learn.

Le Hy's model, based on IOHMMs and the associated learning algorithms, based Laplace's rule of succession and on the IBW, are a very good basis for establishing a model for believable agents in virtual environments. With BIBot, the implementation of Le Hy's model, we were able to confirm some of the strengths of the model and to discover some weaknesses.

Le Hy's model is very flexible and easy to understand. The learning algorithms are well adapted to the model and the goal of believability. However, the way the model handle perceptions and actions may break the illusion of believability. There are also some limitations with the structure of the model, making the addition of sensors difficult and the expression of simple behaviour impossible. Finally, the learning algorithms could be improved to increase the complexity of the behaviours they can handle.

Contents of Chapter 3

Chapter 3

Chameleon: Behaviour Model and Learning Algorithm for Believable Agents

Summary of 3

This chapter describes four enhancements of Le Hy's work to achieve more believable behaviours. We split the sensors into two types representing two granularities of information: high-level stimuli and low-level stimuli to clarify [P2: Sensors]. We also defined two kinds of actions each one associated to a kind of stimuli. Then we propose an attention selection mechanism where the agent selects one high-level stimulus and one low-level which answers to the problems [P4: Scaling] and [P5: Readability]. This mechanism makes the model more flexible and allows the model to express more complex behaviours, solving [P3: Expressiveness]. We present a modification of the Growing Neural Gas (GNG) to learn by imitation information about the layout of the environment giving an answer to [P1: Navigation]. Finally, we propose a revamped imitation learning algorithm to learn almost all the model parameters resolving the problem [P6: Learning].

Introduction of 3

In the previous chapter, we described Le Hy's architecture which consists in the sequencing of decisions using Bayesian programming. This architecture is simple to settle and seems promising according to the preliminary results given by BIBot. The concept of decision makes the behaviour easy to adjust by modifying the distributions. These are the reasons we decided to follows Le Hy's general idea. First, a decision is chosen knowing the previous one and some information about the environment. Then actions are done depending on the current decision and the environment. However some limitations were raised in the previous chapter (see section 2.4.2) so we propose a series of modifications to make the behaviour more believable.

Plan of 3

In order to clarify the model and increase its expressiveness, we propose a semantic refinement of the sensors and actions in section 3.1. In order to compensate for the lack of expressiveness of the FEC and to make the parameters for the IP easier to read, we suggest to replace those two mechanisms by an attention mechanism in section 3.2. For the agent to be able to adapt itself to new environments, we propose a learning algorithm in section 3.3 to imitate the way users move in the virtual environment. Finally, to learn most of the model parameters, we propose an EM imitation learning algorithm in section 3.4.

3.1 Semantic Refinement

Introduction of 3.1

The problem [P2: Sensors] is related to the independence between the stimuli and the other random variables in the model. We found that when implementing the model, decision and actions were obviously independent from some stimuli. Therefore, it is possible to alleviate the work of the learning algorithm by specifying from the beginning the independences. the goal is thus to make the learning faster to achieve [L3: Fast].

Plan of 3.1

We will first split the stimuli into two kinds of random variables in section 3.1.1. One type represents general information and the other type represents accurate information. In section 3.1.2 we will define two kind of random variables for each type of actions, one depending from accurate information the other from the general state of the environment.

3.1.1 Categorization of Stimuli

In is implementation, Le Hy used different kinds of information for the choice of the decision and the actions [P2: Sensors]. In order to pick the

decision D, the implementation uses stimuli giving the value of the agent's remaining life points, the weapon it is holding, etc. (Le Hy et al., 2004). So as to pick the action A_i, the implementation uses stimuli which represent angles to objects (Le Hy, 2007, pages 36–54, in French).

This implementation of the model is logical: the kind of information needed to take a decision and to perform an action is different. Indeed, you only need to know that there is food nearby to decide you want to eat but you need its exact location to grab it. This distinction must be clearly stated in the model with the definition of high and low-level stimuli to be able to point out the consequences of this choice. It will also make easier for the developers to define which stimuli should influence the choice of D^t or A^t.

3.1.1.1 High-Level Stimuli

High-level stimuli represent the global state of the agent and its environment. The random variable H_i^t represents the value of the i^{th} high-level stimulus available in the environment at time t. They will be used to pick the decision D^t and actions only needing this kind of information. We use the following notations:

- N_H the number of high-level stimuli
- $\mathcal{H}_i$ the set of values the i^{th} high-level stimulus can take
- H^t the conjunction of all the high-level stimuli at time t which can be written $H^t = (H_1^t, \ldots, H_{N_H}^t)$
- $\mathcal{H}$ the set of possible values for H^t: $\mathcal{H} = \mathcal{H}_1 \times \ldots \times \mathcal{H}_{N_H}$

High-level stimuli can be used to represent internal information about the agent: hunger, level of hormones, pain, etc. They can also represent global information about the surroundings: temperature, presence of food, lighting, etc.

3.1.1.2 Low-Level Stimuli

Low-level stimuli are accurate information about the agent's surroundings. The random variable L_i^t represents the i^{th} low-level stimulus available in the environment at t. They will be used to pick actions which need accurate data to be performed. We use the following notations:

- N_L the number of low-level stimuli
- $\mathcal{L}_i$ the set of values the i^{th} high-level stimulus can take

- L^t the conjunction of all the low-level stimuli at time t which can be written $L^t = (L^t_1, \ldots, L^t_{N_L})$

- $\mathcal{L}$ the set of possible values for L^t: $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_{N_L}$

Because we consider that our agent is not omniscient, at time t, it knows only about the low-level stimuli with index in $K^t \subset [1, N_L]$, the other ones may not be visible or known.

Low-level stimuli can be used to represent the exact location of surroundings objects, their speed and direction. The summary of the relation between decisions and stimuli can be found in figure 3.1.

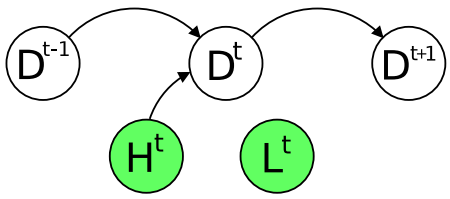


Figure 3.1: *Partial representation of the model depicting only the relation between the decision D^t and the stimuli H^t and L^t.*

In order to make the notations simpler, we note the stimuli the agent considers at time t: $\mathcal{S}^t = (L^t, H^t, K^t)$.

3.1.1.3 Generalization over Stimuli

The number of high and low-level stimuli defined this way may be particularly high. The agent must be able to find an appropriate behaviour when confronted to a stimulus which associated distribution may not have been learned yet. The model must then allow the agent to generalize, making the agent chose the best action in term of believability.

We designed a solution based on the work of Shepard (1987) which consist in defining a distance between stimuli. If stimuli are close using this distance, the behaviour should be approximatively the same. The agent, when facing a stimulus which has not be seen during the learning, will find the closest stimulus and act according to this substitute. It can improve significantly the quality of the behaviour at the beginning of the learning and for very rare situations.

The actual value of the distance is unimportant, the only requirement is that two close stimuli produce *a priori* similar behaviours. The way the distance is computed is to be defined during the implementation. For instance if the agent sees a tennis ball on the ground, it can act as if it was a football ball. Of course, the illusion will not last long but it may fool the players the time needed for the learning. It can also be applied to positions: if the ball is 90° right, it can act as it is 45^{circ} right, the agent may be able to adjust its behaviour the next time step.

3.1.1.4 Human-like Stimuli?

According to the requirement [B6: Perception], the behaviour model should be given stimuli reflecting the information the player have. Indeed, if the model has the same information as a player, it is more likely to display the same behaviours. However giving the exact same stimuli is infeasible because it would mean the model should be plugged to real sensors and sit in front of a real computer.

As this is not a conceivable option, we approximate the player's sensory information by the avatar's. We consider the player has all the information the avatar has, and is looking in the same direction as the avatar. This is an acceptable approximation as all the information is indeed displayed on the player's screen and emitted through his/her speakers. However it is not certain that the player is aware of the information because of the limitation of the human perception mechanisms.

The main drawback of this method is that the information given to the avatar is usually represented in a quite different fashion. As the avatar is in the virtual world it has access to very accurate positions, speed, etc. but does not usually have information about lighting or attenuation of sounds. Each virtual environment has its specificities, but each of the differences between the agent and the player's perception can make the agent act in a unbelievable manner if it takes account information that a player could not have.

3.1.2 Categorization of Actions

In Le Hy's model all the actions are considered independent given the decision and the sensory information. We will keep this hypothesis as it allows to reduce the complexity without making the agent act in a totally unbelievable manner. Indeed, when combining the actions into a complex action, the number of possibilities is way higher. However no real improvement in the behaviour were noticed with this kind of actions.

Since the model has two different kinds of stimuli, the actions can be chosen depending on the level of detail they need to be performed. First, we define actions depending only on the high-level stimuli which usually allow the agent to act on itself. We will call them *reflexive actions*. Then we define actions depending only on the low-level stimuli which usually allow the agent to affect its environment. We will call them *external actions*.

3.1.2.1 Reflexive Actions

A reflexive action models an action the agent uses on itself, to change its status, without modifying the environment. We denote R_i^t the random variable giving the i^{th} reflexive action at time t. Reflexive actions are chosen according to the current decision and the high-level stimuli: R_i^t is picked using the random distribution $\mathbb{P}\left(R_i^t \middle| D^t, H^t\right)$. This models the fact that the agent does not need very accurate information about its surroundings in order to achieve reflexive actions. We use the following notations:

- N_R the number of reflexive actions
- $\mathcal{R}_i$ the set of values the i^{th} reflexive action can take
- R^t the conjunction of all the reflexive actions at time t which can be written $R^t = (R_1^t, \ldots, R_{N_R}^t)$
- $\mathcal{R}$ the set of possible values for R^t: $\mathcal{R} = \mathcal{R}_1 \times \ldots \times \mathcal{R}_{N_R}$

Reflexive actions can be for instance chewing, yawning and most facial expressions. They can also be used to make the agent scratch or pick objects in its pockets although it may depends on the kind of environment the model is used in.

3.1.2.2 External Actions

An external action models an action the agent uses to directly affect its surroundings. We denote E_i^t the random variable giving the i^{th} external action at time t. This kind of action is chosen according to the current decision and the low-level stimuli: E_i^t is randomly picked using the random distribution $\mathbb{P}\left(E_i^t \middle| D^t, L^t\right)$. This kind of action allow the definition of very accurate stimuli to perform accurate actions without adding complexity to the choice of D. Indeed, D is chosen using the high-level stimuli. We use the following notations:

- N_E the number of external actions
- $\mathcal{E}_i$ the set of values the i^{th} external action can take

- E^t the conjunction of all the external actions at time t which can be written $E^t = (E^t_1, \ldots, E^t_{N_E})$

- $\mathcal{E}$ the set of possible values for E^t: $\mathcal{E} = \mathcal{E}_1 \times \ldots \times \mathcal{E}_{N_E}$

External action are designed to model interactions with the environment such as picking, pushing and moving objects. Communications with other players like waving, hugging or shouting can be also used by such actions (due to the complexity of talking, it will not be discussed in this thesis). Finally, as the agent considers the environment in an egocentric way, moving is also considered as an action having an impact on the environment: walking, climbing, turning. Indeed moving changes all the perception of the environment the agent has.

For the Chameleon model, we have the following equality for each time step t: $\mathcal{A}^t = (R^t, E^t)$.

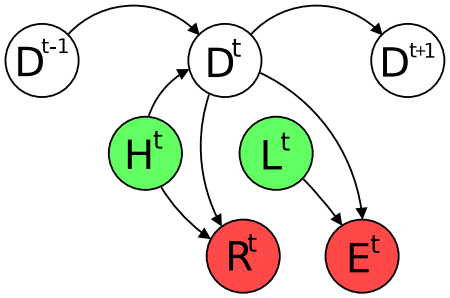


Figure 3.2: *Partial representation of the model depicting the relation between the decision D^t, the stimuli H^t and L^t and the actions R^t and E^t.*

3.1.3 An Example

Throughout this chapter we will use a example to explain our theoretical model represented in figure 3.2. We define the random variables and their values in the table 3.1 and give a graphical representation in figure 3.3).

In order to compare our model to Le Hy's, we define the equivalent model using Le Hy's proposition (see figure 3.4). The independence between some stimuli, decisions and actions reduce the number of parameters of the model. A more detailed analysis is done in the section 4.1.3.

Variable	Definition	Values
	High-level stimuli	
H_1	$FoodInMouth$	$(No, Solid, Chewed)$
H_2	$Hunger$	$(Low, Medium, High)$
	Low-level stimuli	
L_1	$FoodPosition$	$(Close, Far) \times (Right, Left)$
	Reflexive actions	
R_1	$Chew$	(Yes, No)
R_2	$Swallow$	(Yes, No)
	External actions	
E_1	$PickFood$	(Yes, No)
E_2	$Walk$	$(Foward, Backward)$
E_3	$Turn$	$(Right, Left)$
	Decisions	
D	$Decision$	$(FindFood, Eat)$

Table 3.1: *Example of a model following the Chameleon proposition.*

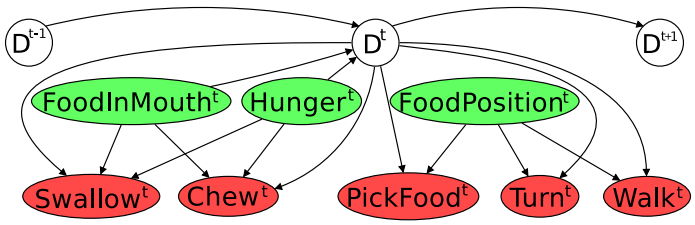


Figure 3.3: *Example of an application of the model. FoodInMouth and Hunger are high-level stimuli, FoodPosition is a low-level stimulus. Chew and Swallow are two reflexive actions. Walk, Turn and PickFood are external actions.*

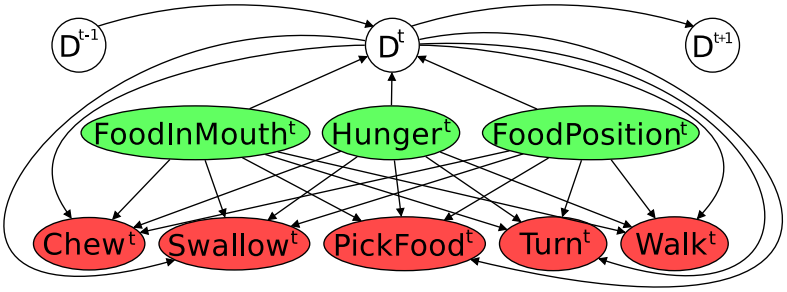


Figure 3.4: *Example of a model following Le Hy's specifications and aiming at expressing the same behaviours as the example in figure 3.3.*

Conclusion of 3.1

Because we found that different level of detail were needed for the choice of decisions and actions, we decided to clarify the model [P2: Sensors]. Two kinds of stimuli were defined, high-level, general information, H_i^t, and low-level, accurate information, L_i^t. We also defined two kinds of action, reflexive, R_i^t, and external actions, E_i^t, each one being a group of dependent actions.

Depending on the level of detail needed for the definition of the behaviours, probability distributions will be defined using H_i^t or L_i^t: decisions and reflexive actions need high-level information while external actions need low-level information (see figure 3.2).

Our model can be seen as a generalization of Le Hy's model. By stating that $H^t = S^t$, the random variables L^t and E^t being unused, our model is equivalent to Le Hy's.

3.2 Attention Selection Mechanism

Introduction of 3.2

The FEC is not able to express complex behaviours [P3: Expressiveness] and the IP makes it difficult to add lots of sensors [P4: Scaling] and is hard to read [P5: Readability]. Both FEC and IP are designed to reduce the complexity of the model. Indeed, expressing directly a distribution $\mathbb{P}(X \,|Z_1, ..., Z_n)$ can be infeasible because it may requires to store too many values. The goal is to find a mechanism to split the problem into distributions of the kind $\mathbb{P}(X \,|Z_i)$.

In order to replace the two mechanisms, we propose an attention selection mechanism. The idea is simple: at each time step t, the agent focus on one high-level stimulus and one low-level stimulus. Although the mechanism is simplistic, it fits perfectly with the goal of believability:

- By focusing on one information, our agent may appear more believable than an agent switching constantly between goals, achieving none of them.
- The users may better understand what the agent wants to do as its behaviour is focused on one specific information [B5: Understandable].
- In case our agent focus on the wrong information, the users may assume that it did not see the interesting information, thus not breaking the illusion.

Plan of 3.2

This attention selection mechanism use two random variables I^t and J^t which give respectively the index of the high-level stimulus and the index of the low-level stimulus the agent focus on. The mechanism defined to choose I^t will be detailed in section 3.2.1 and the one for J^t will be detailed in section 3.2.2.

The following notation are adopted:

- $\mathcal{Q}^t = (I^t, J^t, D^t)$: the hidden state at time t
- $t_b..t_e$ is the interval of time from time t_b to time t_e

3.2.1 High-Level Attention

The random variable I^t is influenced by the high-level stimuli in the following way:

$$\mathbb{P}\left(I^t \left| \mathcal{A}^{1..t}, \mathcal{S}^{1..t}, \mathcal{Q}^{1..t} - I^t \right.\right) = \mathbb{P}\left(I^t \left| H^t \right.\right) \tag{3.1}$$

where $\mathcal{A}^{1..t}$ is the sequence of actions, $\mathcal{S}^{1..t}$ the sequence of stimuli $\mathcal{Q}^{1..t}$ the sequence of hidden variables from time 1 to time t and $\mathcal{Q}^{1..t} - I^t$ is the sequence of hidden variables deprived from the variable I^t. Equation (3.1) accounts only for the dependence between I^t and other random variables. But this conditional distribution requires $N_H \times |\mathcal{H}|$ parameters, which would make our model intractable. Indeed, adding a new high-level stimuli H_p would multiply the number of values for the distribution by $|\mathcal{H}_p|$. Therefore we propose to add some constraints on I^t, assuming that its conditional distribution has a specific shape: we define a function $\theta_i(H_i) \in \mathcal{R}_+^*$ for each high-level stimulus and we impose the following formula:

$$\mathbb{P}\left(I^t = i \left| H^t \right.\right) = \frac{\theta_i\left(H_i^t\right)}{\sum_{n=1}^{N_H} \theta_n\left(H_n^t\right)} \tag{3.2}$$

This formula means that each high-level stimulus has an absolute attention value. The higher the value, the more likely the model will focus on the high-level stimuli. In order to have a probability distribution, the attention is marginalised over all the high-level stimuli. The main assumption which is done by this formula is that the attention of a high-level stimuli does not depend directly on the other stimuli (except for the marginalisation).

For instance, using the same example as in table 3.1, we can define values for θ (see table 3.2). If the agent has no food in mouth and is hungry, it has a probability of $\frac{20}{21}$ of focusing on its hunger and a probability of $\frac{1}{21}$ of focusing on its empty mouth. However if the agent is already eating something (solid or chewed), it has a probability of $\frac{5}{6}$ of focusing on the content of its mouth and a probability of $\frac{1}{6}$ to focus on its hunger.

Stimulus	θ
H_1 $FoodInMouth$	
No	1
Solid	100
Chewed	100
H_2 $Hunger$	
Low	1
Medium	5
High	20

Table 3.2: *Example of attention values for the model given in table 3.1.*

Once the agent has focused on one high-level stimulus, the agent must take a decision as if the stimulus $H^t_{I^t}$ was the only one that he had received, and according to his previous decision D^{t-1}. This is the main reason the high-level attention does not depend on the decision, to avoid the creation of interdependencies. In our model, we assume that the choice of D is done according to the formula:

$$\mathbb{P}\left(D^t \left| \mathcal{A}^{1..t}, \mathcal{S}^{1..t}, \mathcal{Q}^{1..t} - D^t\right.\right) = \mathbb{P}\left(D^t \left| D^{t-1}, H^t, I^t\right.\right) \tag{3.3}$$

$$= \mathbb{P}\left(D^t \left| D^{t-1}, H^t_{I^t}\right.\right) \tag{3.4}$$

We do not need any more the IP hypothesis, allowing us to add many sensors without compromising the accuracy of the results. Moreover, the distribution $\mathbb{P}\left(D^t \left| D^{t-1}, H^t_i\right.\right)$ is easier to understand than its IP counterpart. We still managed to break the complexity, $\mathbb{P}\left(D^t \left| D^{t-1}, H^t_i\right.\right)$ and θ being both tractable.

The decision being chosen, the model can then pick a reflexive action, R^t_u which is driven by the high-level stimulus $H^t_{I^t}$. In our model, the distribution for this choice is assumed to be:

$$\mathbb{P}\left(R^t_u \left| \mathcal{A}^{1..t} - R^t_u, \mathcal{S}^{1..t}, \mathcal{Q}^{1..t}\right.\right) = \mathbb{P}\left(R^t_u \left| D^t, H^t, I^t\right.\right) \tag{3.5}$$

$$\mathbb{P}\left(R^t_u \left| D^t, H^t_{I^t}\right.\right) \tag{3.6}$$

3.2.2 Low-Level Attention

The agent has now to choose an external action. Unlike high-level, low-level attention J^t takes into account the current decision: it depends on D^t and on the known stimuli $\left(L^t_j\right)_{j \in K^t}$ (let us recall that the set of known stimuli at time t is denoted K^t and is a random variable):

$$\mathbb{P}\left(J^t \left| \mathcal{A}^{1..t}, \mathcal{S}^{1..t}, \mathcal{Q}^{1..t} - J^t\right.\right) = \mathbb{P}\left(J^t \left| D^t, L^t, K^t\right.\right) \tag{3.7}$$

As for the variable I^t, the conditional distribution of the equation (3.7) requires too many parameters to be tractable ($N_L|\mathcal{D}||\mathcal{L}|2^{N_L}$), N_L being the number of low-level stimuli in the environment. Therefore, the conditional distribution of J^t has a specific shape:

$$\mathbb{P}\left(J^t = j \,\middle|\, D^t, L^t, K^t\right) = \frac{\mathbb{1}_{K^t}(j)\lambda_j\left(D^t, L^t_j\right)}{\sum_{m=1}^{N_L} \mathbb{1}_{K^t}(m)\lambda_j\left(D^t, L^t_m\right)} \tag{3.8}$$

The formula means that between all the known low-level stimuli, the agent is more likely to choose the low-level stimuli with a high λ. The denominator is only here to make sure the sum of the probabilities is 1.

The agent can then choose the external action E^t_f regarding the stimulus $L^t_{J^t}$. The choice of these action is assumed to follow the distributions:

$$\mathbb{P}\left(E^t_f \,\middle|\, \mathcal{A}^{1..t} - E^t_f, \mathcal{S}^{1..t}, \mathcal{Q}^{1..t}\right) = \mathbb{P}\left(E^t \,\middle|\, D^t, L^t, J^t\right) \tag{3.9}$$

$$= \mathbb{P}\left(E^t_f \,\middle|\, D^t, L^t_{J^t}\right) \tag{3.10}$$

3.2.3 Summary of the Inner Workings of the Model

The model uses a very simple algorithm (see figure 3.5) and has the following parameters:

- the conditional distribution of I^t : $(\theta_i)_{i\in\{1,\dots,N_H\}}$.
- the conditional distribution of J^t : $(\lambda_j)_{j\in\{1,\dots,N_L\}}$.
- the conditional distributions of D^t : $\mathbb{P}\left(D^t = d \,\middle|\, D^{t-1} = d', H^t_i = h_i\right)$, denoted $m_i(d|d', h_i)$. Remember that i is the result of a random pick of I^t.
- the initial conditional distributions of D^1 : $\mathbb{P}\left(D^t = d \,|H_i = h_i\right)$, denoted $m^1_i(d|h_i)$. Here too, i is the result of a random pick of I^t.
- the conditional distribution of R^t_u : $\mathbb{P}\left(R^t_u = r_u \,|D = d, H_i = h_i\right)$, denoted $n_{ui}(r_u|d, h_i)$. Here too, i is the result of a random pick of I^t.
- the conditional distribution of E^t_f: $\mathbb{P}\left(E^t_f = e_f \,\middle|\, D^t = d, L_j = l_j\right)$, denoted $o_{fj}(e_f|d, l_j)$. Remember that j is the result of a random pick of J^t.

The relations between the random variable are summarized in figure 3.6 and a concrete example is given in figure 3.7.

while agent in world **do**
  h ← agent's high-level stimuli
  l ← agent's low-level stimuli
  Pick i using (3.2)
  Pick d using $\mathbb{P}\left(D^t = d \,\middle|\, D^{t-1} = d^{t-1}, H_i^t = h_i\right)$
  for all $u \in [\![1, N_R]\!]$ **do**
    Pick r using $\mathbb{P}\left(R_u^t = r_u \,\middle|\, D^t = d^t, H_i = h_i\right)$
  end for
  Pick j using (3.8)
  for all $f \in [\![1, N_E]\!]$ **do**
    Pick e using $\mathbb{P}\left(E_f^t = e_f \,\middle|\, D^t = d^t, L_j = l_j\right)$
  end for
  Make avatar do $(r_1, ..., r_{N_R}, e_1, ..., e_{N_E})$
  $d^{t-1} \leftarrow d$
end while

Figure 3.5: *Algorithm of the model. It is possible to choose the way the value are picked: randomly following the distribution, using the maximum value in the distribution, etc.*

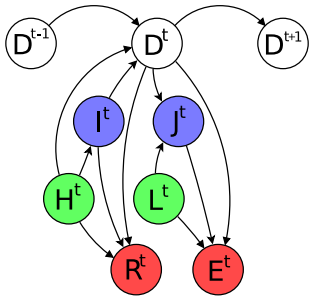


Figure 3.6: *Summary of the relation between the random variable of the model. In green the inputs (sensory data), in red the outputs (actions) and in blue the attention variables.*

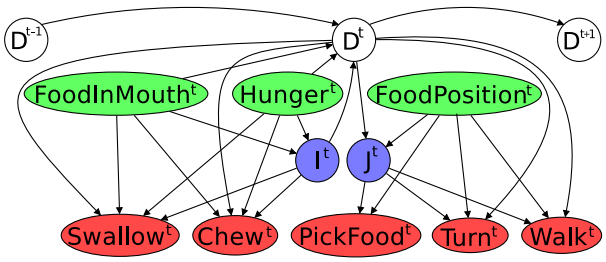


Figure 3.7: *Whole model applied to the example defined in section 3.1.3.*

Conclusion of 3.2

In this section we introduced an attention selection mechanism which consists in making the model focus on one high-level stimulus and one low-level stimulus. This allows the model to avoid the combinatorial explosion for its probability distribution.

In order to define how the model will select the stimuli to focus on, we had to define two sets of functions θ_i and λ_j which give the importance of the information they represent. Indeed, we cannot define directly distributions to give the stimuli to focus on because they would be intractable due to the high number of possibilities.

3.3 Learning the Environment

Introduction of 3.3

The behaviour of a player depends on his/her perceptions of the environment. Similarly, the agent acts depending on its perceptions, but they are different from the players because the agent does not have the same sensors. That is why we have to choose carefully the kind of stimuli we give to the agent, and even better, to make the agent learn to find the best stimuli. It would allow the agent to adapt to new conditions [P1: Navigation]. In this section we choose to focus on the learning of the environment layout because it is one of the harder information to represent for the agent. As we aim at a agent's believable behaviour, we choose to learn the representation of the environment by imitation to include directly in the sensors the way the players act.

Plan of 3.3

First we explain the reasons for choosing an algorithm called Growing Neural Gas (GNG) to learn the representation of the environment and detail its principle in section 3.3.1. Then we introduce some modifications for the algorithm to fit our needs in section 3.3.2. Finally,we explain how the algorithm and the information it gives are integrated in the model in section 3.3.3.

3.3.1 Principle of the Growing Neural Gas (GNG)

Models which control virtual humanoids use different kinds of representation to find paths to go from one point to an other. All the meshes used to render the environment are too complex for agents to handle them. As a consequence, classic approaches use a graph to represent accessible places with nodes and paths between each place by edges (see figure 3.8). Actual solutions tend to use a simple mesh, with different degrees of complexity, to represent the accessible zones (see figure 3.9). The problem with the latter solution is that it requires an algorithm to find the optimal path between two points, path which may not be believable. Moreover, a graph solution is more adapted to

be used in a model similar to Le Hy's: each node of the graph can be used by the model to attract or push back the avatar.

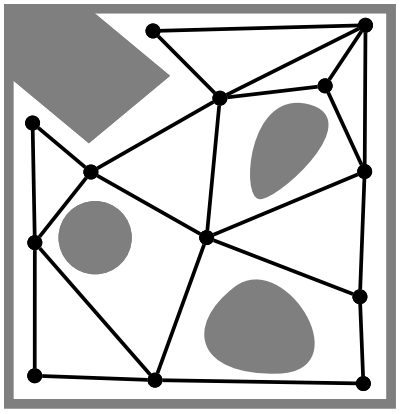


Figure 3.8: *A simple environment (obstacles are in grey) represented by a graph. Nodes are noted by circles and edges by black lines. An avatar can go from one node to an other only if the nodes are connected by an edge. Usually, an A* is used to find the path between two nodes.*

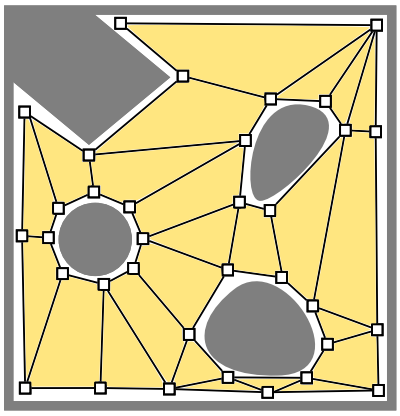


Figure 3.9: *A simple environment (obstacles are in grey) represented by a mesh. The avatar can navigate in the zone defined by the mesh (in yellow) because it knows they are no obstacle in this zone. Different algorithms can be used to find optimal paths.*

In order to achieve the best believability, we want the nodes of the graph to be learned by imitation of a human player instead of being placed by a designer. This work as been done in (Thurau et al., 2004) where nodes and a potential field are learned from humans playing a video game. The agent can then use this representation to move in the game environment, following

the field defined at each node. In order to learn the positions of the nodes, Thurau uses an algorithm called Growing Neural Gas (GNG).

The GNG (Fritzke, 1995) is a graph model which is capable of incremental learning. Each node has a position (x,y,z) in the environment and has a cumulated error which measures how well the node represents its surroundings: the fewer the error of a node, the better the node represents its surroundings. Each edge links two nodes and has an age which gives the time it was last activated. This algorithm needs to be omniscient, because the position of the teacher is to be known at any time.

The principle of the GNG is to modify its graph for each input of the teacher's position in order to fit the graph with the teacher's position. The model can add or remove nodes and edges if they are not representative of the behaviour and change the position of the nodes to better represent the teacher's position. The principle of the GNG is explained in the figures 3.10 and 3.11.

while Number of nodes $\leq N_{max}$ **do**
  Get input position (3.11 (a))
  Pick the closest (n_1) and the second closest nodes (n_2) (3.11 (b))
  Create edge between n_1 and n_2 (3.11 (c)).
  If an edge already existed, reset its age to 0.
  Increase the error of n_1 (3.11 (d))
  Move n_1 and its neighbours toward the input (3.11 (e))
  Increase the age of all the edges emanating from n_1 by 1 (3.11 (f))
  Delete edges exceeding a certain age (3.11 (g))
  if Iteration number is a multiple of η **then**
    Find the maximum error node n_{max}
    Find the neighbour of max with maximum n_{max2} (3.11 (h))
    Insert node between n_{max} and n_{max2} (3.11 (i))
    Decrease the error of n_{max} and n_{max2}
  end if
  Decrease each node's error by a small amount (3.11 (j))
end while

Figure 3.10: *The algorithm of the GNG as defined in (Fritzke, 1995).*

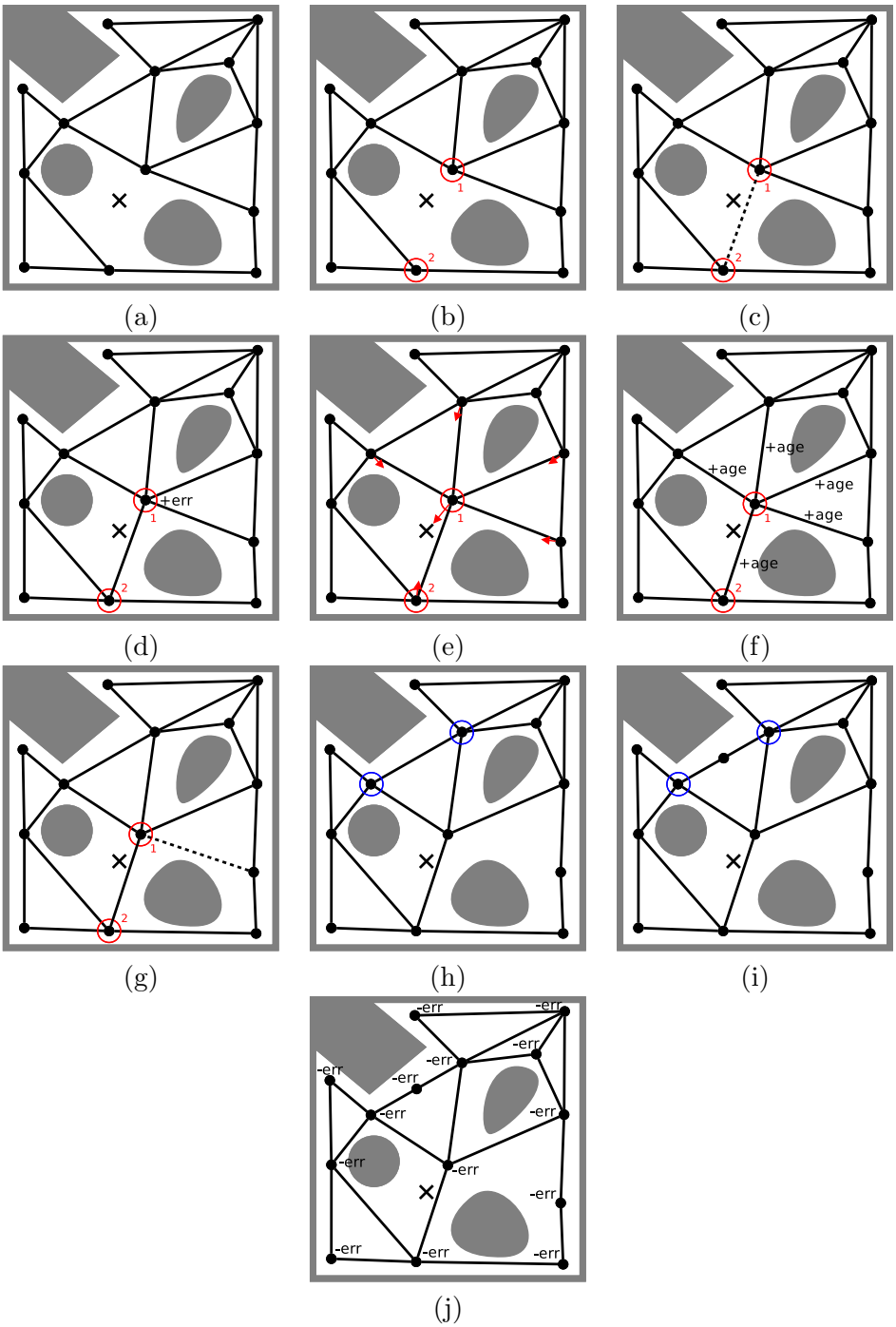


Figure 3.11: *Detail of the steps of the GNG algorithm. The black cross is the input, black circles are the nodes of the GNG and black lines are the edges of the GNG. Gray shapes represent the obstacles in the environment.*

3.3.2 Modification of the Growing Neural Gas (GNG)

The GNG version we use is modified as shown by figure 3.12. The main problem with the GNG is that we do not know when the learning is finished, if it ever finishes. We propose that instead of inserting a new node each η input, a node is inserted when the error of a node is superior to a parameter $\overline{Err}$. As each node's error is reduced by a small amount $\grave{E}\acute{r}r$ for each input, the modified GNG algorithm does not need a stopping criterion. Indeed, if there are many nodes which represent well the environment, the error added for the input will be small and for a set of inputs, the total added error will be distributed among several nodes. The decreasing of error will avoid new nodes to be added to the GNG resulting in a stable state. However, if the teacher goes to a place in the environment he/she has never been before, the added error will be high enough to counter the decay of the error, resulting in the creation of new nodes.

This algorithm has five parameters which influence the density of nodes, the quality of the representation, the adaptivity and the convergence time:

- The attraction $\overrightarrow{attract_1}$ applied to n_1 toward (x, y, z)
- The attraction $\overrightarrow{attract_2}$ applied to the neighbours of n_1 toward (x, y, z)
- The error decay for nodes, $\grave{E}\acute{r}r$
- The maximum error for nodes, $\overline{Err}$
- The maximum age for the edges, $\overline{Age}$

3.3.3 Integration in the Model and Learning Algorithm

The nodes learned by this model can be used directly by the model as low-level stimuli. Indeed, they represent precise information which will be particularly important for the choice of motion actions. However, the information given by the edges cannot be used as it denotes only proximity between nodes and not a path between them. Two nodes may be linked by an edge because they are close with a obstacle between them.

This modified version of the GNG should begin to learn before the main learning algorithm. Indeed, as the nodes are used during the learning of the model parameters, incorrect placement or insufficient nodes may cause the main learning algorithm to learn incorrect probability distributions. However, as our GNG should not be stopped, in case the teacher changes its behaviour, the two learnings processes can be executed in parallel.

```
nodes ← {}
edges ← {}
while teacher plays do
  (x,y,z) ← teacher's position
  if |nodes| = 0 or 1 then
    nodes ← nodes ∪ {(x,y,z,error=0)}
  end if
  if |nodes| = 2 then
    edges ← {(nodes,age=0)}
  end if
  n1 ← closest((x,y,z),nodes)
  n2 ← secondClosest((x,y,z),nodes)
  edge ← edges ∪ {{n1,n2},age=0)}

  n1.error+=||(x,y,z)-n1||
  Attract n1 toward (x,y,z)
  ∀ edge ∈ edgesFrom(n1), edge.age++
  Delete edges older than Age
  Attract neighbours(n1) toward (x,y,z)
  ∀ node ∈ nodes, node.error-=Err

  if n1.error > Err then
    maxErrNei ← maxErrorNeighbour(n1)
    newNode ← between(n1,maxErrNei)
    n1.error/=2
    maxErrNei.error/=2
    newError ← n1.error+maxErrNei.error
    nodes ← nodes ∪ {(newNode,newError)}
  end if
end while
```

Figure 3.12: *Algorithm used to learn the topology of the environment represented by a GNG.*

This should not be a problem in term of resources because the GNG has a complexity of $O(n)$ where n is the number of nodes in the graph. This should leave plenty of computing power for the other learning algorithm. The algorithm is totally capable of handling data form several teachers. By giving inputs from several teachers we should increase a lot the speed of the learning. The only pitfall with this technique is that teachers can have totally different usages of the environment. As a consequence, the learned GNG may not

reflect neither of the two behaviours.

Conclusion of 3.3

In this section we presented a modified version of an algorithm, the GNG, for the imitation learning of the topology of virtual environments. The algorithm is able to learn from one or more teachers how to use the environment by representing it with a graph. As the nodes are placed where the teachers go, this should give a representation allowing the learning and the generation of believable behaviours. However, the information gathered and given to the agent is limited to the nodes of the graph, coordinates in space.

3.4 Learning the Model Parameters via an EM Algorithm

Introduction of 3.4

With the GNG algorithm, our agent is able to learn some of the stimuli needed to move in the environment. We can now focus on the learning of the model parameters. In his work, Le Hy uses a modified version of the IBW (Florez-Larrahondo, 2005). We prefer not to use this algorithm because it makes very simplistic assumptions (see section 2.3.2.3) [P6: Learning]. Therefore we will revamp the whole learning algorithm by applying the EM technique to our problem, avoiding assumptions incompatible with our goal of believability.

Plan of 3.4

We first present the theoretical principle of the EM and its application to our model in section 3.4.1. Then we detail the two steps of the EM. First the expectation procedure in section 3.4.2 where the algorithm estimates the probabilities of the model to generate an observed behaviour and then the maximization procedure in section 3.4.3 where the model parameters are updated to fit the observed behaviour better.

3.4.1 Expectation-Maximization algorithm (EM)

The theoretical principle of the EM algorithm was first introduced in (Dempster et al., 1977) and its convergence was fully proven in (Wu, 1983). It has been applied to a wide range of problems where the data is said to be incomplete. Incomplete data means that only a part of data is known, the other part is to be guessed in order to learn on a pseudo-complete data.

In our case, the algorithm gathers the values of L_j^t, H_i^t, the stimuli, and R^t, E^t, the actions, at each time step. The values of I^t, J^t and D^t, the hidden states, are not known, thus the data is incomplete. We will apply an EM algorithm to be able to learn the model parameters.

The following notation are adopted:

- T is the length of the sequence of observation used for the learning
- $\mathcal{A} = \mathcal{A}^{1..T}$, the total sequence of observed actions
- $\mathcal{S} = \mathcal{S}^{1..T}$, the total sequence of observed stimuli
- $\mathcal{Q} = \mathcal{Q}^{1..T}$, the total sequence of hidden states
- Φ is the set of model parameters

Throughout this section we will use a simple example to illustrate the meaning of the estimators (see figure 3.13). This example consist in a sequence of observation lasting 5 time steps where the values of $\mathcal{A}$ and $\mathcal{S}$ are known. the values in $\mathcal{Q}$ are unknown and must be guessed knowing the current model parameters, $\mathcal{A}$ and $\mathcal{S}$.

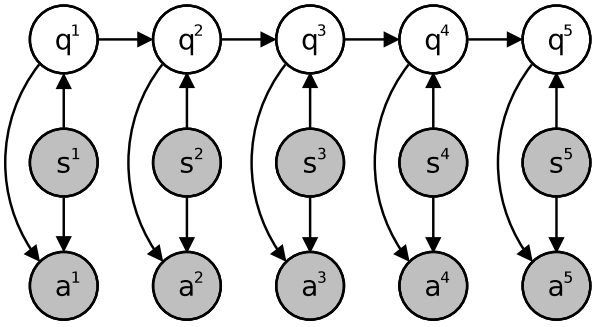


Figure 3.13: *A simple example of the relation between the values of the random variables of the model. Gray filled circles represent the observed values on a teacher. White filled circles represent the hidden values. This diagram depicts a whole sequence of observation which lasts 5 time steps.*

Our goal is to find the best model parameters, Φ^*, such as the likelihood, $P(\mathcal{A}|\mathcal{S}, \Phi^*)$, is maximal. This means we want to find the parameters of the model the most likely to generate an observed sequence given the sensory information: if the model were in place of the observed player, it would most likely generate approximatively the same actions. As we do not have any information about the hidden variables ($\mathcal{Q}$) we choose to use an EM to find a local maximum. The idea is to find iteratively a Φ_{n+1} such that $P(\mathcal{A}|\mathcal{S}, \Phi_{n+1}) \geq P(\mathcal{A}|\mathcal{S}, \Phi_n)$.

In the sequel, we will not maximize the likelihood, but the log-likelihood $L(\mathcal{A}|\mathcal{S},\Phi_n) = \log P(\mathcal{A}|\mathcal{S},\Phi_n)$. This is equivalent as log is strictly increasing. The reason to do so is that it will allow some simplification in the maximization problem. However, the function is not easy to maximize, we need to simplify the problem (this is also a part of the proof of convergence):

$$\begin{aligned} L(\mathcal{A}|\mathcal{S},\Phi) &= \log \mathbb{P}\left(\mathcal{A}\,|\mathcal{S},\Phi\right) \\ &= \log \sum_{\mathcal{Q}} \mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right) && (3.11) \\ &= \log \left(\sum_{\mathcal{Q}} \frac{\mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right)}{\mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right)} \mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right) \right) && (3.12) \end{aligned}$$

Because *log* is concave we have:

$$\geq \sum_{\mathcal{Q}} \mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right) \log \left(\frac{\mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right)}{\mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right)} \right) \quad (3.13)$$

Then we can compute the lower bound of $L(\mathcal{A}|\mathcal{S},\Phi)$ in the following way:

$$\begin{aligned} &\sum_{\mathcal{Q}} \mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right) \log \left(\frac{\mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right)}{\mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right)} \right) \\ &= \sum_{\mathcal{Q}} \mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right) \log \left(\mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right) \right) \\ &\quad - \sum_{\mathcal{Q}} \mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right) \log (\mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right)) && (3.14) \\ &= Q(\Phi|\Phi_n) + R(\Phi_n|\Phi_n) && (3.15) \end{aligned}$$

thus, we have:

$$L(\mathcal{A}|\mathcal{S},\Phi) \geq Q(\Phi|\Phi_n) + R(\Phi_n|\Phi_n) \quad (3.16)$$

This inequality partly proves that by choosing Φ_{n+1} such that $\Phi_{n+1} = \text{argmax}_{\Phi}\, Q(\Phi|\Phi_n)$, the log-likelihood will converge toward a local maximum of $L(\mathcal{A}|\mathcal{S},\Phi)$. The problem is now to maximize $Q(\Phi|\Phi_n)$. First we need to find how $\mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right)$ can be expressed:

$$\begin{aligned} &\mathbb{P}\left(\mathcal{A},\mathcal{Q}\,|\mathcal{S},\Phi\right) \\ &= \mathbb{P}\left(\mathcal{A}\,|\mathcal{Q},\mathcal{S},\Phi\right) \mathbb{P}\left(\mathcal{Q}\,|\mathcal{S},\Phi\right) && (3.17) \\ &= \prod_{t=1}^{T} \mathbb{P}\left(\mathcal{A}^t\,|\mathcal{Q},\mathcal{S},\Phi\right) \mathbb{P}\left(\mathcal{Q}\,|\mathcal{S},\Phi\right) && (3.18) \\ &= \prod_{t=1}^{T} \mathbb{P}\left(\mathcal{A}^t\,|\mathcal{Q}^t,\mathcal{S}^t,\Phi\right) \mathbb{P}\left(\mathcal{Q}^1\,|\mathcal{S},\Phi\right) \prod_{t=2}^{T} \mathbb{P}\left(\mathcal{Q}^t\,|\mathcal{Q}^{1..t-1},\mathcal{S},\Phi\right) && (3.19) \end{aligned}$$

$$= \prod_{t=1}^{T} \mathbb{P}\left(R^t, E^t \left| I^t, J^t, D^t, \mathcal{S}^t, \Phi\right.\right) \mathbb{P}\left(\mathcal{Q}^1 \left| \mathcal{S}^1, \Phi\right.\right) \prod_{t=2}^{T} \mathbb{P}\left(\mathcal{Q}^t \left| \mathcal{Q}^{t-1}, \mathcal{S}^t, \Phi\right.\right) \quad (3.20)$$

$$= \prod_{t=1}^{T} \mathbb{P}\left(R^t \left| I^t, D^t, \mathcal{S}^t, \Phi\right.\right) \mathbb{P}\left(E^t \left| J^t, D^t, \mathcal{S}^t, \Phi\right.\right)$$

$$\mathbb{P}\left(I^1, J^1, D^1 \left| \mathcal{S}^1, \Phi\right.\right) \prod_{t=2}^{T} \mathbb{P}\left(I^t, J^t, D^t \left| I^{t-1}, J^{t-1}, D^{t-1}, \mathcal{S}^t, \Phi\right.\right) \quad (3.21)$$

$$= \prod_{t=1}^{T} n_i(r^t|d^t, h_{i^t}^t) o_j(e^t|d^t, l_{j^t}^t) \mathbb{P}\left(I^t \left| \mathcal{S}^t, \Phi\right.\right) \mathbb{P}\left(J^t \left| D^t, \mathcal{S}^t, \Phi\right.\right)$$

$$\mathbb{P}\left(D^1 \left| \mathcal{S}^1, \Phi\right.\right) \prod_{t=2}^{T} \mathbb{P}\left(D^t \left| I^t, D^{t-1}, \mathcal{S}^t, \Phi\right.\right) \quad (3.22)$$

$$= \prod_{t=1}^{T} n_i(r^t|d^t, h_{i^t}^t) o_j(e^t|d^t, l_{j^t}^t) \frac{\theta_{i^t}(h_{i^t}^t)}{\sum_{x=1}^{N_H} \theta_x(h_x^t)} \frac{\mathbb{1}_{k^t}(j^t)\lambda_{j^t}\left(d^t, l_{j^t}^t\right)}{\sum_{y=1}^{N_L} \mathbb{1}_{k^t}(y)\lambda_y\left(d^t, l_m^t\right)}$$

$$m_i^1(d^1|h_{i^1}^t) \prod_{t=2}^{T} m_i(d^t|d^{t-1}, h_{i^t}) \quad (3.23)$$

Using this last result we can then express $Q(\Phi, \Phi_n)$. The advantage of using the *log* function becomes obvious here, because it allows us to split the function into sums:

$$Q(\Phi, \Phi_n) = \sum_{\mathcal{Q}} \log\left(\mathbb{P}\left(\mathcal{A}, \mathcal{Q} \left| \mathcal{S}, \Phi\right.\right)\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.24)$$

$$= \sum_{\mathcal{Q}} \sum_{t=1}^{T} \log\left(n_i(r^t|d^t, h_{i^t}^t)\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.25)$$

$$+ \sum_{\mathcal{Q}} \sum_{t=1}^{T} \log\left(o_j(e^t|d^t, l_{j^t}^t)\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.26)$$

$$+ \sum_{\mathcal{Q}} \sum_{t=1}^{T} \log\left(\frac{\mathbb{1}_{k^t}(j^t)\lambda_{j^t}\left(d^t, l_{j^t}^t\right)}{\sum_{y=1}^{N_L} \mathbb{1}_{k^t}(y)\lambda_y\left(d^t, l_y^t\right)}\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.27)$$

$$+ \sum_{\mathcal{Q}} \sum_{t=1}^{T} \log\left(\frac{\theta_{i^t}(h_{i^t}^t)}{\sum_{x=1}^{N_H} \theta_x(h_x^t)}\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.28)$$

$$+ \sum_{\mathcal{Q}} \sum_{t=2}^{T} \log\left(m_i(d^t|d^{t-1}, h_{i^t})\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.29)$$

$$+ \sum_{\mathcal{Q}} \log\left(m_i^1(d^1|h_{i^1}^t)\right) \mathbb{P}\left(\mathcal{Q} \left| \mathcal{A}, \mathcal{S}, \Phi_n\right.\right) \quad (3.30)$$

Using our notations and because some of the previous quantities may depend on only a few of the hidden states, one has:

$$Q(\Phi, \Phi_n) = \sum_{i,d} \sum_{t=1}^{T} \log \left[n_i(r^t | d, h_{i^t}^t) \right] \mathbb{P}\left(i^t = i, d^t = d \,|\mathcal{A}, \mathcal{S}, \Phi_n \right) \tag{3.31}$$

$$+ \sum_{d,j} \sum_{t=1}^{T} \log \left[o_j(e^t | d^t, l_{j^t}^t) \right] \mathbb{P}\left(d^t = d, j^t = j \,|\mathcal{A}, \mathcal{S}, \Phi_n \right) \tag{3.32}$$

$$+ \sum_{d,j} \sum_{t=1}^{T} \log \left[\frac{\mathbb{1}_{k^t}(j) \lambda_j \left(d, l_{j^t}^t \right)}{\sum_{y=1}^{N_L} \mathbb{1}_{k^t}(y) \lambda_y \left(d^t, l_y^t \right)} \right] \mathbb{P}\left(d^t = d, j^t = j \,|\mathcal{A}, \mathcal{S}, \Phi_n \right) \tag{3.33}$$

$$+ \sum_{i} \sum_{t=1}^{T} \log \left[\frac{\theta_i \left(h_i^t \right)}{\sum_{x=1}^{N_H} \theta_x \left(h_x^t \right)} \right] \mathbb{P}\left(i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi_n \right) \tag{3.34}$$

$$+ \sum_{i,d,d'} \sum_{t=2}^{T} \log \left[m_i(d | d', h_i^t) \right] \mathbb{P}\left(d^{t-1} = d', i^t = i, d^t = d \,|\mathcal{A}, \mathcal{S}, \Phi_n \right) \tag{3.35}$$

$$+ \sum_{i,d} \log \left[m_i^1(d | h_i^1) \right] \mathbb{P}\left(i^1 = i, d^1 = d \,|\mathcal{A}, \mathcal{S}, \Phi_n \right) \tag{3.36}$$

We have now detailed the shape of the function to be maximized. Our learning problem is then reduced to an optimization problem, like in most machine learning. By maximizing the function $Q(\Phi, \Phi_n)$, the algorithm will find the parameters the most likely to generate the observed behaviour. But, before addressing the maximization problem, we need to give an explicit shape for some terms that will be used in the sequel.

3.4.2 Expectation Procedure

In the maximization procedure, the algorithm will need the probability for the model to be in specific hidden state configurations. For instance, the likelihood of being in the hidden state $I^t = i, D^t = d$ which can be written $\mathbb{P}\left(i^t = i, d^t = d \,|\mathcal{A}, \mathcal{S}, \Phi_n \right)$, will be used in the maximization formulae. In order to compute all these probabilities, we will use an approach inspired by the forward backward technique. This method decreases the complexity of the computations by defining probabilities in an iterative manner. The reason we

did not use the classic forward backward technique is that it uses probabilities which would look like $\mathbb{P}\left(\mathcal{A}, i^t = i, d^t = d \,|\mathcal{S}, \Phi_n\right)$ which are often too small to be handled by a computer.

Let us define the following forward variables (we recall that $q = (i, j, d)$). They are called "forward" because they only use information from time 1 to time t (see figure 3.14):

$$\alpha_q^t = \mathbb{P}\left(q^t = q \,|\mathcal{A}^{1..t}\mathcal{S}\Phi\right) \tag{3.37}$$

$$\alpha_{d,j}^t = \mathbb{P}\left(d^t = d, j^t = j \,|\mathcal{A}^{1..t}\mathcal{S}\Phi\right) \tag{3.38}$$

$$= \sum_i \alpha_q^t \tag{3.39}$$

$$\alpha_{d,i}^t = \mathbb{P}\left(d^t = d, i^t = i \,|\mathcal{A}^{1..t}\mathcal{S}\Phi\right) \tag{3.40}$$

$$= \sum_j \alpha_q^t \tag{3.41}$$

$$\alpha_d^t = \mathbb{P}\left(d^t = d \,|\mathcal{A}^{1..t}\mathcal{S}\Phi\right) \tag{3.42}$$

$$= \sum_{i,j} \alpha_q^t \tag{3.43}$$

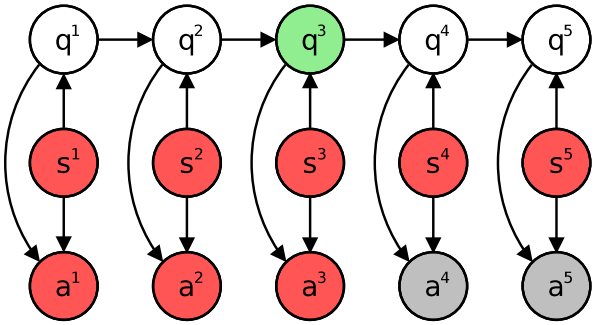


Figure 3.14: *Illustration of the meaning of the estimator α. In this figure $\alpha_{q^3}^3$ is represented which estimates the probability of the green hidden value given the observed red values. White and grey values are unknown.*

This quantity may be defined recursively (we dropped the symbol Φ for the sake of simplicity):

$$\alpha_q^1 = \frac{\mathbb{P}\left(\mathcal{A}^1 \,|q^1 = q, S\right)\mathbb{P}\left(q^1 = q \,|S\right)}{\mathbb{P}\left(\mathcal{A}^1 \,|S\right)} \tag{3.44}$$

$$\propto n_i(r^1|d, h_i^1) m(e^1|d, l_j^1) \frac{\mathbb{1}_{k^1}(j)\lambda_j(d, l_j^1)}{\sum_y \mathbb{1}_{k^1}(y)\lambda_y(d, l_y^1)} m_i^1(d|h_i^1) \frac{\theta_i(h_i^1)}{\sum_x \theta_x(h_x^1)} \tag{3.45}$$

The recursion formula is ($\frac{1}{Z}$ being a normalization factor which can be computed over q):

$$\alpha_q^{t+1} = \sum_{q'} \mathbb{P}\left(q^{t+1} = q, q^t = q' \left| \mathcal{A}^{1..t+1}, \mathcal{S}\right.\right) \tag{3.46}$$

$$= \sum_{q'} \frac{\mathbb{P}\left(q^{t+1} = q, q^t = q', \mathcal{A}^{t+1} \left| \mathcal{A}^{1..t}, \mathcal{S}\right.\right)}{\mathbb{P}\left(\mathcal{A}^{t+1} \left| \mathcal{A}^{1..t}, \mathcal{S}\right.\right)} \tag{3.47}$$

$$= \sum_{q'} \frac{1}{Z} \mathbb{P}\left(q^{t+1} = q, \mathcal{A}^{t+1} \left| q^t = q', \mathcal{A}^{1..t}, \mathcal{S}\right.\right) \mathbb{P}\left(q^t = q' \left| \mathcal{A}^{1..t}, \mathcal{S}\right.\right) \tag{3.48}$$

$$= \frac{1}{Z} \sum_{q'} \mathbb{P}\left(q^{t+1} = q, \mathcal{A}^{t+1} \left| q^t = q', \mathcal{S}\right.\right) \alpha_{q'}^t \tag{3.49}$$

$$= \frac{1}{Z} \sum_{q'} \mathbb{P}\left(\mathcal{A}^{t+1} \left| q^{t+1} = q, q^t = q', \mathcal{S}\right.\right) \mathbb{P}\left(q^{t+1} = q \left| q^t = q', \mathcal{A}^{1..t}, \mathcal{S}\right.\right) \alpha_{q'}^t \tag{3.50}$$

$$= \frac{1}{Z} \mathbb{P}\left(\mathcal{A}^{t+1} \left| q^{t+1} = q, \mathcal{S}\right.\right) \sum_{q'} \mathbb{P}\left(q^{t+1} = q \left| q^t = q', \mathcal{S}\right.\right) \alpha_{q'}^t \tag{3.51}$$

$$= \frac{1}{Z} n_i(r^{t+1}|d, h_i^{t+1}) o(e^{t+1}|d, l_j^{t+1}) \frac{\mathbb{1}_{k^{t+1}}(j)\lambda_j(d, l_j^{t+1})}{\sum_y \mathbb{1}_{k^{t+1}}(y)\lambda(d, l_y^{t+1})} \frac{\theta_i(h_i^{t+1})}{\sum_x \theta_x(h_x^{t+1})} \sum_{d'} m_i(d|d', h_i^{t+1}) \alpha_{d'}^t \tag{3.52}$$

Similarly, we define the backward variable. They are called backward because they use information from time T to time $t+1$ (see figure 3.15):

$$\beta_q^t = \frac{\mathbb{P}\left(q^t = q \left| \mathcal{A}^{t+1..T}, \mathcal{S}, \Phi\right.\right)}{\mathbb{P}\left(q^t = q \left| \mathcal{S}, \Phi\right.\right)} \tag{3.53}$$

$$\beta_d^t = \frac{\mathbb{P}\left(d^t = d \left| \mathcal{A}^{t+1..T}, \mathcal{S}, \Phi\right.\right)}{\mathbb{P}\left(d^t = d \left| \mathcal{S}, \Phi\right.\right)} \tag{3.54}$$

This quantity may be defined recursively (again we dropped the symbol Φ for the sake of simplicity):

$$\beta_q^T = 1 \tag{3.55}$$

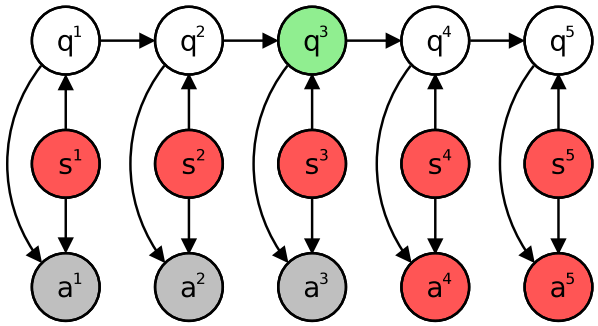


Figure 3.15: *Illustration of the meaning of the estimator β. In this figure $\beta^3_{q^3}$ is represented which estimates the probability of the green hidden value given the observed red value. White and grey values are unknown.*

The recursion formula is ($\frac{1}{Z}$ being a normalization factor which can be computed over q):

$$\beta_q^{t-1} = \frac{\mathbb{P}\left(q^{t-1}=q \,|\mathcal{A}^{t..T},\mathcal{S}\right)}{\mathbb{P}\left(q^{t-1}=q\,|\mathcal{S}\right)} \tag{3.56}$$

$$= \sum_{q'} \frac{\mathbb{P}\left(q^{t-1}=q, q^t=q' \,|\mathcal{A}^{t..T},\mathcal{S}\right)}{\mathbb{P}\left(q^{t-1}=q\,|\mathcal{S}\right)} \tag{3.57}$$

$$= \sum_{q'} \frac{\mathbb{P}\left(\mathcal{A}^t, q^{t-1}=q, q^t=q' \,|\mathcal{A}^{t+1..T}\mathcal{S}\right)}{\mathbb{P}\left(\mathcal{A}^t\,|\mathcal{A}^{t+1..T}\mathcal{S}\right)\mathbb{P}\left(q^{t-1}=q\,|\mathcal{S}\right)} \tag{3.58}$$

$$= \frac{1}{Z}\sum_{q'} \frac{\mathbb{P}\left(\mathcal{A}^t, q^{t-1}=q \,|q^t=q', \mathcal{A}^{t+1..T},\mathcal{S}\right)\mathbb{P}\left(q^t=q' \,|\mathcal{A}^{t+1..T},\mathcal{S}\right)}{\mathbb{P}\left(q^{t-1}=q\,|\mathcal{S}\right)} \tag{3.59}$$

$$= \frac{1}{Z}\sum_{q'} \frac{\mathbb{P}\left(\mathcal{A}^t, q^{t-1}=q \,|q^t=q', \mathcal{S}\right)\mathbb{P}\left(q^t=q' \,|\mathcal{A}^{t+1..T},\mathcal{S}\right)}{\mathbb{P}\left(q^{t-1}=q\,|\mathcal{S}\right)} \tag{3.60}$$

$$= \frac{1}{Z}\sum_{q'} \frac{\mathbb{P}\left(\mathcal{A}^t, q^{t-1}=q, q^t=q' \,|\mathcal{S}\right)}{\mathbb{P}\left(q^{t-1}=q\,|\mathcal{S}\right)}\frac{\mathbb{P}\left(q^t=q' \,|\mathcal{A}^{t+1..T},\mathcal{S}\right)}{\mathbb{P}\left(q^t=q'\,|\mathcal{S}\right)} \tag{3.61}$$

$$= \frac{1}{Z}\sum_{q'} \mathbb{P}\left(\mathcal{A}^t, q^t=q' \,|q^{t-1}=q,\mathcal{S}\right)\beta^t_{q'} \tag{3.62}$$

$$= \frac{1}{Z}\sum_{q'} n_i(r^t|d',h^t_{i'})o_j(e^t|d',l^t_{j'})\frac{\mathbb{1}_{k^t}(j')\lambda_{j'}(d',l^t_{j'})}{\sum_y \mathbb{1}_{k^t}(y)\lambda_y(d',l^t_y)}\frac{\theta_{i'}(h^t_{i'})}{\sum_x \theta_x(h^t_x)} \\ m_i(d'|d,h^t_{i'})\beta^t_{q'} \tag{3.63}$$

As we can see, β^t_q depends only on d, so we have $\beta^t_q = \beta^t_d$.

The forward and backward variables are very useful because they can be defined recursively. Now that we defined them, we can use them to express the probabilities we will need in the maximization procedure.

We introduce

$$\gamma_q^t = \mathbb{P}\left(q^t = q \,|\mathcal{A}, \mathcal{S}\right) \tag{3.64}$$

which is the probability of begin in the hidden state q given all the observations (see figure 3.16).

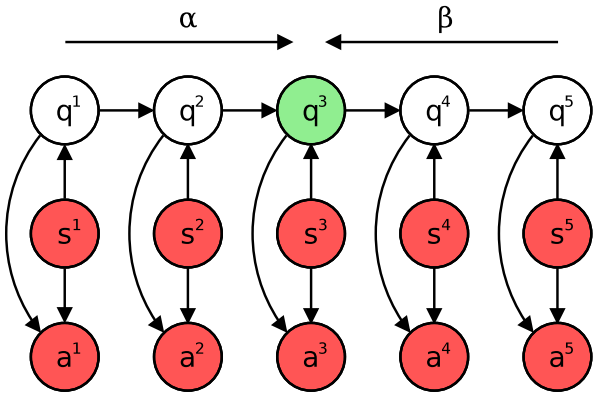


Figure 3.16: *Illustration of the meaning of the estimator γ. In this figure $\gamma_{q^3}^3$ is represented which estimates the probability of the green hidden value given the observed red values. White values are unknown.*

Then we can write:

$$\begin{aligned}
\gamma_q^t &= \mathbb{P}\left(q^t = q \,|\mathcal{A}, \mathcal{S}\right) && (3.65)\\
&= \frac{\mathbb{P}\left(\mathcal{A}, q^t = q \,|\mathcal{S}\right)}{\mathbb{P}\left(\mathcal{A} \,|\mathcal{S}\right)} && (3.66)\\
&= \frac{\mathbb{P}\left(\mathcal{A}, q^t = q \,|\mathcal{S}\right)}{\sum_{q'} \mathbb{P}\left(\mathcal{A}, q^t = q' \,|\mathcal{S}\right)} && (3.67)\\
&= \frac{\mathbb{P}\left(\mathcal{A}^{1..t}, q^t = q \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,|q^t = q, \mathcal{A}^{1..t}, \mathcal{S}\right)}{\sum_{q'} \mathbb{P}\left(\mathcal{A}^{1..t}, q^t = q' \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,|q^t = q', \mathcal{A}^{1..t}, \mathcal{S}\right)} && (3.68)\\
&= \frac{\mathbb{P}\left(\mathcal{A}^{1..t}, q^t = q \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,|q^t = q, \mathcal{S}\right)}{\sum_{q'} \mathbb{P}\left(\mathcal{A}^{1..t}, q^t = q' \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,|q^t = q', \mathcal{S}\right)} && (3.69)\\
&= \frac{\mathbb{P}\left(q^t = q \,|\mathcal{A}^{1..t}, \mathcal{S}\right) \frac{\mathbb{P}\left(q^t = q | \mathcal{A}^{t+1..T}, \mathcal{S}\right)}{\mathbb{P}(q^t = q | \mathcal{S})}}{\sum_{q'} \mathbb{P}\left(q^t = q' \,|\mathcal{A}^{1..t}, \mathcal{S}\right) \frac{\mathbb{P}(q^t = q' | \mathcal{A}^{t+1..T}, \mathcal{S})}{\mathbb{P}(q^t = q' | \mathcal{S})}} \frac{\mathbb{P}\left(\mathcal{A}^{1..t} \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,|\mathcal{S}\right)}{\mathbb{P}\left(\mathcal{A}^{1..t} \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,|\mathcal{S}\right)} && (3.70)
\end{aligned}$$

$$= \frac{\alpha_q^t \beta_d^t}{\sum_{q'} \alpha_{q'}^t \beta_{d'}^t} \tag{3.71}$$

$$\propto \alpha_q^t \beta_d^t \tag{3.72}$$

Similarly, we define an estimator for partial configurations of hidden states:

$$\gamma_{d,i}^t = \mathbb{P}\left(d^t = d, i^t = i \,|\mathcal{A}, \mathcal{S}\right) \tag{3.73}$$

$$= \frac{\alpha_{d,i}^t \beta_d^t}{\sum_{d',i'} \alpha_{d',i'}^t \beta_{d'}^t} \tag{3.74}$$

$$\gamma_{d,j}^t = \mathbb{P}\left(d^t = d, j^t = j \,|\mathcal{A}, \mathcal{S}\right) \tag{3.75}$$

$$= \frac{\alpha_{d,j}^t \beta_d^t}{\sum_{d',j'} \alpha_{d',j'}^t \beta_{d'}^t} \tag{3.76}$$

$$\gamma_i^t = \mathbb{P}\left(i^t = i \,|\mathcal{A}, \mathcal{S}\right) \tag{3.77}$$

$$= \sum_d \gamma_{d,i}^t \tag{3.78}$$

We also need to define the probability of a transition between two hidden states (see figure 3.17):

$$\xi_{q',q}^t = \mathbb{P}\left(q^{t-1} = q', q^t = q \,|\mathcal{A}, \mathcal{S}, \Phi\right) \tag{3.79}$$

$$= \frac{\mathbb{P}\left(\mathcal{A}, q^{t-1} = q', q^t = q \,|\mathcal{S}\right)}{\mathbb{P}\left(\mathcal{A} \,|\mathcal{S}\right)} \tag{3.80}$$

$$= \frac{1}{Z}\mathbb{P}\left(\mathcal{A}^{1..t-1}, q^{t-1} = q' \,|\mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t..T}, q^t = q \,\middle|\, \mathcal{A}^{1..t-1}, q^{t-1} = q', \mathcal{S}\right) \tag{3.81}$$

$$= \frac{1}{Z}\alpha_{q'}^{t-1} \mathbb{P}\left(\mathcal{A}^{t..T}, q^t = q \,\middle|\, q^{t-1} = q', \mathcal{S}\right) \tag{3.82}$$

$$= \frac{1}{Z}\alpha_{q'}^{t-1} \mathbb{P}\left(\mathcal{A}^{t..T} \,\middle|\, q^t = q, q^{t-1} = q', \mathcal{S}\right) \mathbb{P}\left(q^t = q \,\middle|\, q^{t-1} = q', \mathcal{S}\right) \tag{3.83}$$

$$= \frac{1}{Z}\alpha_{q'}^{t-1} \mathbb{P}\left(\mathcal{A}^t, \mathcal{A}^{t+1..T} \,\middle|\, q^t = q, \mathcal{S}\right) \mathbb{P}\left(q^t = q \,\middle|\, q^{t-1} = q', \mathcal{S}\right) \tag{3.84}$$

$$= \frac{1}{Z}\alpha_{q'}^{t-1} \mathbb{P}\left(\mathcal{A}^t \,\middle|\, q^t = q, \mathcal{S}\right) \mathbb{P}\left(\mathcal{A}^{t+1..T} \,\middle|\, q^t = q, \mathcal{S}\right) \mathbb{P}\left(q^t = q \,\middle|\, q^{t-1} = q', \mathcal{S}\right) \tag{3.85}$$

$$= \frac{1}{Z}\alpha_{q'}^{t-1} n_i(r^t|d, h_i^t) o_j(e^t|d, l_j^t) \frac{\mathbb{1}_{k^t}(j)\lambda_j(d, l_j^t)}{\sum_y \mathbb{1}_{k^t}(y)\lambda_y(d, l_y^t)} \frac{\theta_i(h_i^t)}{\sum_x \theta_x(h_x^t)} \beta_q^t m_i(d|d', h_i^t) \tag{3.86}$$

Similarly, we define an estimator for partial configurations of hidden states:

$$\xi_{d',d,i}^t = \mathbb{P}\left(d^{t-1} = d', d^t = d, i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi\right) \tag{3.87}$$

$$= \sum_{i',j',j} \mathbb{P}\left(q^{t-1} = q', q^t = q \,|\mathcal{A}, \mathcal{S}, \Phi\right) \tag{3.88}$$

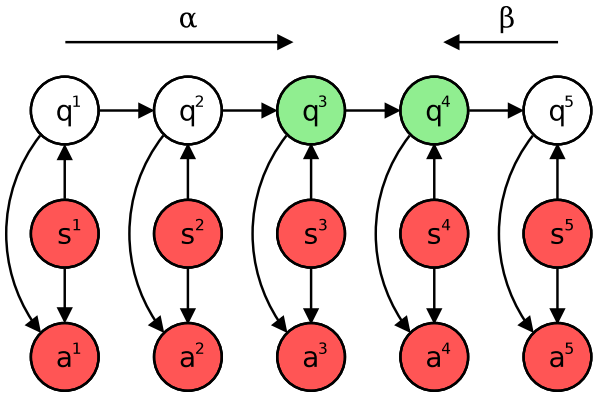


Figure 3.17: *Illustration of the meaning of the estimator ξ. In this figure $\xi^4_{q^3,q^4}$ is represented which estimates the probability of the green hidden value given the observed red values. White values are unknown.*

$$= \sum_{i',j',j} \xi^t_{q',q} \tag{3.89}$$

$$\xi^t_{d',i} = \mathbb{P}\left(d^{t-1} = d', i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi\right) \tag{3.90}$$

$$= \sum_{i',j',j,d} \mathbb{P}\left(q^{t-1} = q', q^t = q \,|\mathcal{A}, \mathcal{S}, \Phi\right) \tag{3.91}$$

$$= \sum_{i',j',j,d} \xi^t_{q',q} \tag{3.92}$$

3.4.3 Maximization Procedure

In order to find a Φ_{n+1} such that $P(\mathcal{A}|\mathcal{S}, \Phi_{n+1}) \geq P(\mathcal{A}|\mathcal{S}, \Phi_n)$, we have to maximize each part of $Q(\Phi, \Phi_n)$, the equations (3.31), (3.32), (3.33), (3.34), (3.35) and (3.36). This is possible because each part of Q is independent from the others. Each maximization procedure will need the estimators γ and ξ defined in the previous section.

3.4.3.1 Maximizing the Quantity (3.31)

We want to maximize the following quantity by finding the optimal distribution $n'_{i^t}(r^t|d^t, h^t_{i^t})$:

$$\sum_{\mathcal{Q}} \sum_{t=1}^{T} \log\left[n_i(r^t|d^t, h^t_{i^t})\right] \mathbb{P}\left(\mathcal{Q} \,|\mathcal{A}, \mathcal{S}, \Phi_n\right)$$

$$= \sum_{i,d} \sum_{t=1}^{T} \log\left[n_i(r^t|d, h^t_i)\right] \mathbb{P}\left(i^t = i, d^t = d \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \tag{3.93}$$

Because they are probability distributions, the conditions one has are:

$$\sum_r n_i(r|d,h_i) = 1 \tag{3.94}$$

The goal is to maximize (3.31) by finding the best parameters, $b'_i(r,d,h_i)$. Using Lagrange multipliers in a classic way leads to:

$$n'_i(r,d,h_i) = \frac{\sum_{t=1}^T \mathbb{P}\left(i^t = i, d^t = d \,|\mathcal{A},\mathcal{S},\Phi_n\right) \mathbb{1}_{h_i^t=h_i, r^t=r}}{\sum_{t=1}^T \mathbb{P}\left(i^t = i, d^t = d \,|\mathcal{A},\mathcal{S},\Phi_n\right) \mathbb{1}_{h_i^t=h_i}} \tag{3.95}$$

$$= \frac{\sum_{t=1}^T \gamma_{d,i}^t \mathbb{1}_{h_i^t=h_i, r^t=r}}{\sum_{t=1}^T \gamma_{d,i}^t \mathbb{1}_{h_i^t=h_i}} \tag{3.96}$$

3.4.3.2 Maximizing the Quantity (3.32)

We want to maximize the following quantity by finding the optimal distribution $o'_j(e^t|d^t,l_{j^t}^t)$:

$$\begin{aligned} &\sum_q \sum_{t=1}^T \log\left[o_j(e^t|d^t,l_{j^t}^t)\right] \mathbb{P}\left(\mathcal{Q}\,|\mathcal{A},\mathcal{S},\Phi_n\right) \\ =&\sum_{j,d} \sum_{t=1}^T \log\left[o_j(e^t|d,l_j^t)\right] \mathbb{P}\left(j^t = j, d^t = d\,|\mathcal{A},\mathcal{S},\Phi_n\right) \end{aligned} \tag{3.97}$$

Because they are probability distributions, the conditions one has are:

$$\sum_r o_j(e|d,l_j) = 1$$

The goal is to maximize (3.32) by finding the best parameters, $c'_j(e,d,l_j)$. Using Lagrange multipliers in a classic way leads to:

$$o'_j(e,d,l_j) = \frac{\sum_{t=1}^T \mathbb{P}\left(j^t = j, d^t = d \,|\mathcal{A},\mathcal{S},\Phi_n\right) \mathbb{1}_{l_j^t=l_j, e^t=e}}{\sum_{t=1}^T \mathbb{P}\left(j^t = j, d^t = d \,|\mathcal{A},\mathcal{S},\Phi_n\right) \mathbb{1}_{l_j^t=l_j}} \tag{3.98}$$

$$= \frac{\sum_{t=1}^T \gamma_{d,j}^t \mathbb{1}_{l_j^t=l_j, e^t=e}}{\sum_{t=1}^T \gamma_{d,j}^t \mathbb{1}_{l_j^t=l_j}} \tag{3.99}$$

3.4.3.3 Maximizing the Quantity (3.33) and (3.34)

We want to maximize the following quantity by finding the optimal distribution $\lambda'_j\left(d, l^t_{j^t}\right)$:

$$\sum_{d,j}\sum_{t=1}^{T}\log\left[\frac{\mathbb{1}_{k^t}(j)\lambda_j\left(d, l^t_j\right)}{\sum_{y=1}^{N_L}\mathbb{1}_{k^t}(y)\lambda_y\left(d, l^t_y\right)}\right]\mathbb{P}\left(d^t = d, j^t = j \,|\mathcal{A}, \mathcal{S}, \Phi_n\right)$$

and we also want to maximize the following quantity by finding the optimal distribution $\theta'_i\left(h^t_i\right)$:

$$\sum_{i}\sum_{t=1}^{T}\log\left[\frac{\theta_i\left(h^t_i\right)}{\sum_{x=1}^{N_H}\theta_x\left(h^t_x\right)}\right]\mathbb{P}\left(i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right)$$

Currently, no solution has been found to maximize efficiently (3.33) and (3.34). Those two quantities correspond to the likelihood of the function λ and θ. A classic gradient descent has been tested to no avail, making the model parameters converge toward values where the agent focus on trivial information.

For now, the value of the two attention function are set by the character designer. Tips are given on how to find good values in sections 3.4.4.2 and 4.2.1. This requires a good knowledge of the environment and the behaviours of the players.

3.4.3.4 Maximizing the Quantity (3.35)

We want to maximize the following quantity by finding the optimal distribution $m'_{i^t}(d^t|d^{t-1}, h^t_{i^t})$:

$$\begin{aligned}&\sum_{q}\sum_{t=2}^{T}\log\left[m_{i^t}(d^t|d^{t-1}, h^t_{i^t})\right]\mathbb{P}\left(\mathcal{Q} \,|\mathcal{A}, \mathcal{S}, \Phi_n\right)\\ =&\sum_{i,d,d'}\sum_{t=2}^{T}\log\left[m_i(d|d', h^t_i)\right]\mathbb{P}\left(d^{t-1} = d', d^t = d, i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right)\end{aligned} \tag{3.100}$$

The goal is to maximize (3.35) by finding the best parameters, $a'_i(d|d', h_i)$. The critical points must satisfy:

$$m'_i(d|d', h_i) = \frac{\sum_t \mathbb{P}\left(d^{t-1} = d', d^t = d, i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \mathbb{1}_{h_i^t = h_i}}{\sum_t \mathbb{P}\left(d^{t-1} = d', i^t = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \mathbb{1}_{h_i^t = h_i}} \tag{3.101}$$

$$= \frac{\sum_t \xi^t_{d',d,i} \mathbb{1}_{h_i^t = h_i}}{\sum_t \xi^t_{d',i} \mathbb{1}_{h_i^t = h_i}} \tag{3.102}$$

3.4.3.5 Maximizing the Quantity (3.36)

We want to maximize the following quantity by finding the optimal distribution $(m^1_{i^1})'(d^1|h^1_{i^1})$:

$$\begin{aligned} &\sum_q \log\left[m^1_{i^1}(d^1|h^1_{i^1})\right] \mathbb{P}\left(\mathcal{Q} \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \\ =&\sum_{i,d} \log\left[m^1_i(d|h^1_i)\right] \mathbb{P}\left(d^1 = d, i^1 = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \end{aligned} \tag{3.103}$$

The goal is to maximize (3.36) by finding the best parameters, $(a^1_i)'(d|h_i)$. Critical points satisfy:

$$(m^1_i)'(d|h_i) = \frac{\mathbb{P}\left(d^1 = d, i^1 = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \mathbb{1}_{h_i^1 = h_i}}{\mathbb{P}\left(i^1 = i \,|\mathcal{A}, \mathcal{S}, \Phi_n\right) \mathbb{1}_{h_i^1 = h_i}} \tag{3.104}$$

$$= \frac{\gamma^1_{d,i} \mathbb{1}_{h_i^1 = h_i}}{\gamma^1_i \mathbb{1}_{h_i^1 = h_i}} \tag{3.105}$$

3.4.4 Putting Expectation and Maximization together

Now that the expectation and the maximization procedures have been detailed, we can explain how to use them in the whole EM.

3.4.4.1 Finding a Sequence of Observations

During the introduction of the algorithm we defined the sequences of observation $\mathcal{A}$ and $\mathcal{S}$ of length T. They are the sequences of what the teacher do and perceive. These sequences allow the model to estimate the probability of the hidden states. As the model is Markovian, the chain of event is very important: too short sequences would lead to erroneous estimations of the probabilities of hidden states. On the opposite, too long sequences will make the algorithm very slow without adding relevant information.

As we applied our model and learning algorithm to a video game and more precisely to a FPS, the choice was a bit more easy. Indeed, in such games, avatars "die" disappearing from the environment and "resurrect" reappearing

in a random place. Therefore a sequence is the actions and stimuli from the "resurrection" to the "death" of the teacher's avatar. Such sequences usually last from 10 seconds to 5 minutes which is not too long for the algorithm.

A last problem concerning the sequences had to be solved: the teacher has a reaction time. Among all the solutions we tried, the most simple worked the best. Instead of associating the actions at t to the stimuli at t, we use the actions at $t + t_{reac}$ms. The actual value of t_{reac} is discussed in 4.4.1.1.

3.4.4.2 Parameters Initialization and Stopping Criterion

The initialization of the parameters is a very important part (Biernacki et al., 2003). A good initialization allows fast convergence and increase the chances of converging toward a local maximum which is close to the global maximum. Finding such initialization for the parameters often includes a quick estimation of their values according to the observations.

We choose to stick to the simplest method for the moment: the random initialization. Each parameter is initialized to a random value, then they are normalized for the sum of probability distributions to be equal to 1. It allows the algorithm to cover many possible solutions at the cost of convergence time.

However, the attention functions, θ and λ, are not randomly initialized because they are not learned. We have to choose their value using our experience. A good way to find good values is to code simple behaviours manually and see the influence of the attention function on the result. As a rule of thumb, other avatars are the most important and objects in the centre of the screen have a higher attention than objects on the border.

The algorithm needs a stopping criterion, based on the quality of the current set of parameters Φ_n. As the algorithm converges toward a local maximum, we do not know *a priori* the value of the likelihood function at this maximum. We have observed that, as the value of $Q(\Phi_{n+1}|\Phi_n)$ converges, the increase is smaller and smaller at each iteration of the algorithm. We based the stopping criterion on this increase: if $Q(\Phi_{n+1}|\Phi_n) - Q(\Phi_n|\Phi_{n-1}) < w$, the algorithm is stopped and Φ_{n+1} us considered to be the solution.

3.4.4.3 Merging the Results of Expectation-Maximization algorithms (EMs)

Like all the EMs, our algorithm is offline, which is contrary to our objective (see section 2.2.3.2). Indeed, the algorithm gives a set of parameters which only satisfies the observed sequence. However, because we have short learning sequences, we can learn from them one after an other, merging the final

resulting set of parameters to a global one, Φ_g. We must keep in mind that each result of the EM is optimal for a single sequence but we want a global solution, acceptable for all sequences.

This solution has however a major weakness because the algorithm discovers the meaning of each decision by specifying the associated distributions. Two running of the algorithm may discover different way to express the behaviour, the decisions from the first run being not comparable to the decisions from the second run. This is specially true when the parameters are learned on different sequences where the teacher has potentially expressed different behaviours.

A first solution is to compute the distance between decisions by using the distributions giving the actions:

$$dist(d, d') = \sum_i \left(n_i(r^t|d, h_i^t) - n_i(r^t|d', h_i^t)\right)^2 + \sum_j \left(o_j(e^t|d, l_j^t) - o_j(e^t|d', l_j^t)\right)^2 \tag{3.106}$$

The two closest decisions are then considered the same, then again with the remaining decisions and again.... This solution gave poor results because the decisions may not be comparable at all. There is also the danger of all distances to be of the same order because of the *curse of dimensionality*.

A better solution is to initialize the learning algorithm with parameters close to the global parameters. We do not use the global parameters directly because there may be zeros in the parameters which is not well handled by EM for initialization. By using the global parameters, the algorithm will use the same "meaning" for each decision as the global ones. This does not guarantee that the decisions will have the same meaning but there are better chances that they will be close.

For the actual merging of the parameters, we consider that the more the observed stimuli, the better the learning. We note $\mathcal{S}_i$ the sequence of stimuli of length T_i used for the learning i. Thus when the learning i is achieved, the distribution v which gives the probability of the random variable z given the stimuli s is updated using the following formula:

$$v(z|s, \Phi_g') = \frac{v(z|s, \Phi_i) \sum_{t=1}^{T_i} \mathbb{1}_{\mathcal{S}_i^t}(s) + v(z|s, \Phi_g) \sum_j \sum_{t=1}^{T_j} \mathbb{1}_{\mathcal{S}_j^t}(s)}{\sum_{t=1}^{T_i} \mathbb{1}_{\mathcal{S}_i^t}(s) + \sum_j \sum_{t=1}^{T_j} \mathbb{1}_{\mathcal{S}_j^t}(s)} \tag{3.107}$$

$i \leftarrow 0$
while teacher plays **do**
  while teacher's avatar alive **do**
    $\mathcal{S}_i^t \leftarrow$ teacher's stimuli
    $\mathcal{A}_i^t \leftarrow$ teacher's actions
    Wait Δt
  end while
  while $Q(\Phi_{i-1}|\Phi_{i-2}) - Q(\Phi_{i-2}|\Phi_{i-3}) < w$ **do**
    Compute α using (3.45), (3.52), (3.43), (3.41) and (3.39)
    Compute β using (3.55) and (3.63)
    Compute γ using (3.72), (3.74), (3.76), and (3.78)
    Compute ξ using (3.86), (3.89) and (3.92)
    Update Φ_i using (3.96), (3.99), (3.102) and (3.105)
  end while
  $i \leftarrow i + 1$
  Update Φ_g using (3.107)
end while

Figure 3.18: *Algorithm used for the imitation learning of the model parameters.*

Conclusion of 3.4

We applied and detailed an EM which allows the learning by imitation of most of the model parameters. The length of the sequence of observation being variable, the algorithm may need a unknown number of iteration to converge correctly, the longer the sequence, the more the iterations. Therefore, we have introduced a classic stopping criterion.

In order to combine the results obtained by the different EM, we used a merging technique to allow the agent to learn the whole behaviour of a player in an online learning algorithm fashion. The whole learning algorithm is written in pseudo-code in figure 3.18.

Conclusion of 3

In this chapter we proposed four modifications to Le Hy's model and learning algorithms. First we defined low-level and high-level stimuli to make the distributions more compact. We grouped actions into two types each one depending of a type of stimulus.

Then we introduced an attention selection mechanism with two sets of functions θ_i and λ_j. These function describes how much the model should pay attention to respectively each high-level stimulus and low-level stimulus. This design avoids intractable amount of parameters.

For the stimuli to reflect the players' behaviour, we added an algorithm to learn by imitation the layout of the environment. By giving such representation to the model, we expect an increase of the believability of the generated behaviours.

Finally, we redesigned the whole learning algorithm from scratch. Our learning algorithm is based on the EM technique, allowing to learn almost all the parameters by imitation. The attention functions θ_i and λ_j are not yet learned. Our algorithm is much slower than Le Hy's but by avoiding simplistic hypothesis, should give much more accurate results.

Contents of Chapter 4

Chapter 4

Analysis and Evaluation of an Implementation of Chameleon

Summary of 4

In this chapter, we present how we adapted our model to the video game UT2004 and the result for each of the four proposed modifications. The semantic refinement of the model reduces the number of parameters for the model making the learning faster and the model clearer. The attention selection mechanism allows the agent to express behaviour which cannot be expressed by Le Hy's model. The GNG makes the agent able to adapt rapidly to unknown environments by observing multiple teachers. Finally, the imitation learning algorithm allows the agent to evolve rapidly toward a more believable behaviour. However, the agent is not able yet to fool players for long.

Introduction of 4

In the previous chapter we proposed four modifications to the Le Hy's work. They aim at making the model able to express and learn more complex behaviours without increasing the complexity of the model. The final goal is to produce more believable behaviours. We must validate our modifications with measures, test cases and evaluations. We will also explain how the model is integrated in the video game UT2004 for our results to be reproducible and to give an idea of how the model can be implemented.

Plan of 4

The organization of this chapter is identical to the previous one. Section 4.1 presents the choice of stimuli and actions to explain how the model can be adapted to a video game. We also analyse the gain of the semantic refinement. Then we present how to choose the value for the attention functions and show a concrete example of how it improves results in section 4.2. In the section 4.3 we analyse the time evolution of the GNG and the impact of the different parameters on the learning. Finally, we try to assess the believability of the behaviours of agents learning by imitation with our algorithm in section 4.4.

4.1 Semantic Refinement

Introduction of 4.1

The semantic refinement of Le Hy's model consists in the partition of the stimuli in two groups: high level ones representing general information and low level ones representing accurate information. The actions are then categorized in two distinct types: reflexive actions which are done by the agent on itself and external actions which allows the agent to interact directly with the environment. The goal of this refinement is to reduce the number of parameters of the model by adding some *a priori* knowledge. The reduction of the number of parameters aims at speeding up the learning, making our agent able of evolving rapidly [B10: Fast Evolution].

Plan of 4.1

We first describe how we plugged our model to the game UT2004: the discretization and categorization of the stimuli is explained in section 4.1.1 and the actions are detailed in section 4.1.2. We then study the effect of the semantic refinement on the number of parameters in section 4.1.3: the impact of each hypothesis introduced by Le Hy and by ourselves is measured for a concrete example.

4.1.1 Choice of the Stimuli

The choice of the high level and low level stimuli is essential to make the agent behave correctly in the environment. Too few information will make the agent miss important events and thus will make the agent behave in an ill-adapted manner. Too much information will make the learning difficult because there will be too many parameters.

4.1.1.1 High Level Stimuli

The high level stimuli represent global information which will allow the agent to take decisions and reflexive actions. We used six different high level stimuli, based on the choices made in (Le Hy, 2007, page 60, in French) and information usually used in UT2004. They are also information available to the player, on its screen, fulfilling the requirement [B6: Perception]:

- Life: gives the amount of hit points the agent's avatar have.
- Number of enemies: gives the number of visible enemies.
- Current weapon: gives the weapon currently in the the hand of the avatar of the agent.
- Current weapon ammunition: gives the current number of ammunition in the weapon in the hand of the avatar of the agent.
- Take damage: tells the agent if it took damage the last time step.
- Weapons in inventory: tells which weapons are in the possession of the avatar.

4.1.1.2 Low Level Stimuli

Low level stimuli represent accurate information which will be used by the agent to choose the interaction and motion actions. We choose a principle of *point of interest*, each low level stimulus being a point in the virtual environment which is important for the agent. We used five different types of points of interest which represent all the important entities in the environment plus information for navigation:

- Player: gives the position of an other player's avatar.
- Weapon: gives the position of a weapon lying on the ground.
- Health: gives the position of an item restoring the avatar's amount of hit points.
- Navigation point: gives the position of a GNG node.
- Ray impact: gives the position of a ray tracing impact.

The first three stimuli (player, weapon and health) represent information available to the player, fulfilling the requirement [B6: Perception]. The last two stimuli (navigation point and ray impact) are not available to the player. They represent information about the topology of the environment which is difficult to represent in a compact way (see section 3.3.1).

The discretization of the position is done in a different way for each of the five types of point of interest but follows the same principle. An example of such discretisation for a *weapon* is given in Figure 4.1 and 4.2. The position for such point of interest is a triple $(pitch, yaw, distance)$.

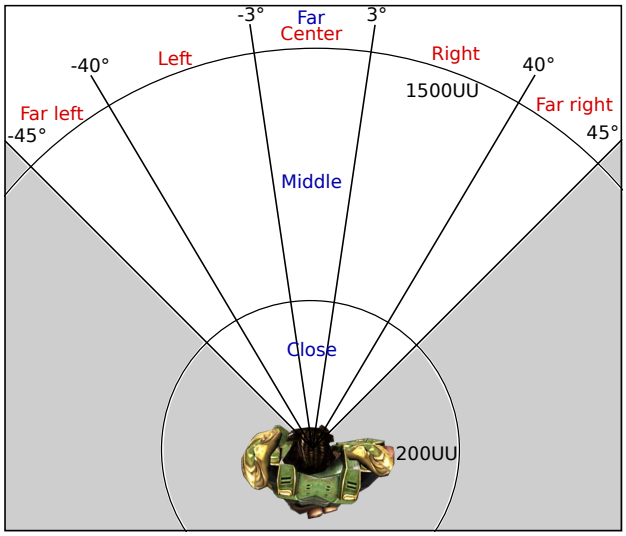


Figure 4.1: *Zones defined for the points of interest of type weapon, top view. The grey zone is outside the Field Of View (FOV). Red values are for yaw, blue for distance. The values in Unreal Unit (UU) gives the distance value between each discretization and the angles in degrees for the yaw.*

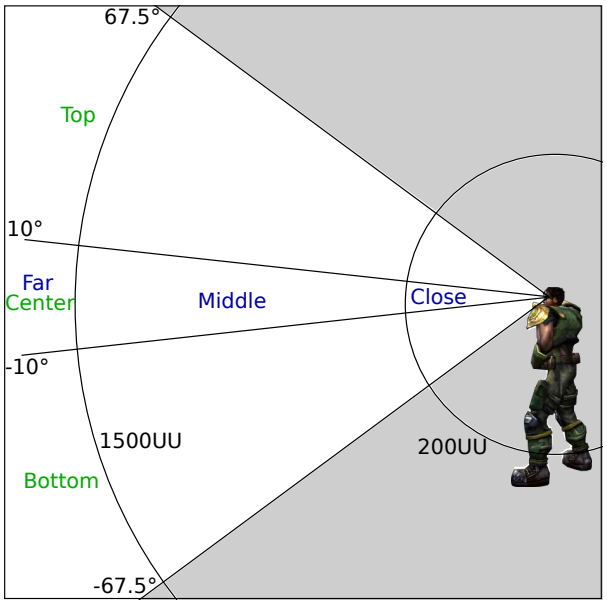


Figure 4.2: *Zones defined for the points of interest of type weapon, side view. The grey zone is outside the FOV. Green values are for pitch, blue for distance. The values in UU gives the distance value between each discretization and the angles in degrees for the pitch.*

4.1.2 Choice of Actions

Reflexive Actions The only reflexive action is an action consisting in witching the weapon currently held in hand with an other weapon the agent is carrying. This action does not involve the environment excepted the agent's avatar. As there are twelve weapons in the environment, the value for this action can take one of this twelve possibilities.

External Actions As UT2004 is based on speed and accuracy, most of the actions need low-level stimuli, thus are external actions. One actions allows the agent to "interact" with the other players by shooting on them. As weapons can be used in two different ways the possible values for this action are: do nothing, pull the trigger or pull the trigger for the alternate mode of the weapon. The principal actions allows the agent to move and aim at other players. We defined five actions: jump, pitch, yaw, forward/backward and lateral movement (see Figure 4.3). We choose these actions because they are very close to what a player can make his/her avatar do in the environment with his/her keyboard and mouse.

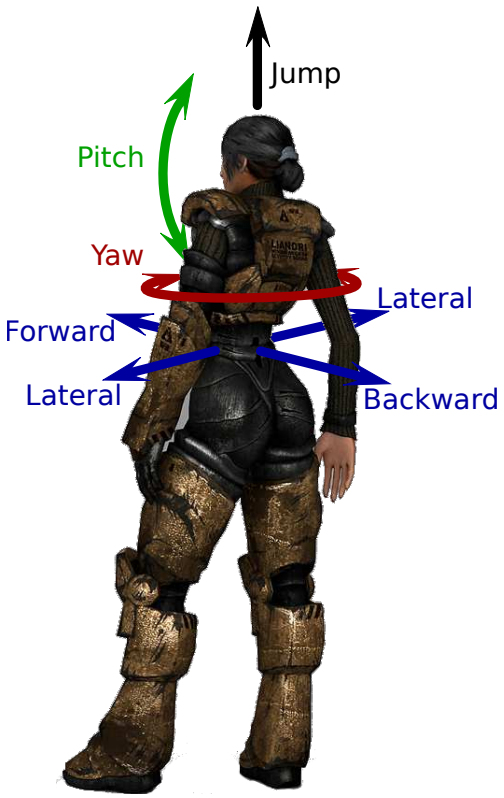


Figure 4.3: *Actions allowing the agent to move in the environment and aim at players.*

4.1.3 Consequences of the Semantic Refinement

In order to study the consequences of the semantic refinement on a concrete example, we will use our application. The definition of each random variable and the number of values they can take is given in Table 4.1.

Variable	Definition	Number of values
	High-level stimuli	
H_1	$Life$	3
H_2	$NumberOfEnemies$	4
H_3	$CurrentWeapon$	14
H_4	$CurrentWeaponAmmunition$	4
H_5	$TakeDamage$	2
H_6	$WeaponsInInventory$	70
	Low-level stimuli	
L_1	$Player$	405
L_2	$Weapon$	45
L_3	$Health$	45
L_4	$NavigationPoint$	72
L_5	$RayImpact$	5
	Reflexive actions	
R_1	$ChangeWeapon$	12
	External actions	
E_1	$Fire$	3
E_2	$Jump$	3
E_3	$Pitch$	3
E_4	Yaw	5
E_5	$Walk$	3
E_6	$Lateral$	3
	Decisions	
D	$Decision$	around 10

Table 4.1: *Definition of each random variable used in the model applied to the game UT2004.*

With this example we can now study the number of values needed for the definition of Le Hy's model and Chameleon. In order to focus only on the influence of the semantic refinement of the stimuli, we will not consider for now the mechanisms to decrease the complexity of the models: FEC and IP for Le Hy's model and attention selection for Chameleon.

Because all the stimuli, decisions and actions are dependent, Le Hy's model would need a total of:

$$\left(\prod_{i=1}^{N_H}|\mathcal{H}_i|\prod_{j=1}^{N_L}|\mathcal{L}_j|\right)|D|^2 \tag{4.1}$$

values for the definition of $\mathbb{P}\left(D^t\left|D^{t-1},\mathcal{S}^t\right.\right)$, the distribution to choose the decision. It would also require

$$|D|\left(\prod_{i=1}^{N_H}|\mathcal{H}_i|\prod_{j=1}^{N_L}|\mathcal{L}_j|\right)\sum_{n=1}^{N_A}|\mathcal{A}_n| \tag{4.2}$$

values for the definition of $\mathbb{P}\left(A_n\left|D^t,\mathcal{S}^t\right.\right)$, the distributions for the choice of the actions.

Chameleon would need a total of

$$\left(\prod_{i=1}^{N_H}\mathcal{H}_i\right)|D|^2 \tag{4.3}$$

values for the definition of $\mathbb{P}\left(D^t\left|D^{t-1},\mathcal{H}^t\right.\right)$, the distribution to choose the decision. It would also require

$$\left(\prod_{i=1}^{N_H}|\mathcal{H}_i|\sum_{u=1}^{N_R}|R_u|+\prod_{j=1}^{N_L}|\mathcal{L}_j|\sum_{f=1}^{N_E}|\mathcal{E}_f|\right)|D| \tag{4.4}$$

values for the definition of the distributions $\mathbb{P}\left(R_u^t\left|D^t,\mathcal{H}^t\right.\right)$ and $\mathbb{P}\left(E_f^t\left|D^t,\mathcal{L}^t\right.\right)$.

The definition of high and low-level stimuli allows the model to reduce the parameters by

$$\left(\prod_{j=1}^{N_L}\mathcal{L}_j\right)|D|^2+\left(\prod_{i=1}^{N_H}|\mathcal{H}_i|\sum_{f=1}^{N_E}|\mathcal{E}_f|+\prod_{j=1}^{N_L}|\mathcal{L}_j|\sum_{u=1}^{N_R}|R_u|\right)|D| \tag{4.5}$$

So the more complex the model, the more favorable is the semantic refinement.

In our application the gain is worthwhile. Indeed, the number of parameters is approximatively divided by 10^5 compared to a model without semantic refinement as show in figure 4.4 and table 4.2. In the game UT2004, the number of states, which approximatively corresponds to decisions in our model, is around 10. For 10 decisions the number of parameters is divided by 1.4×10^5.

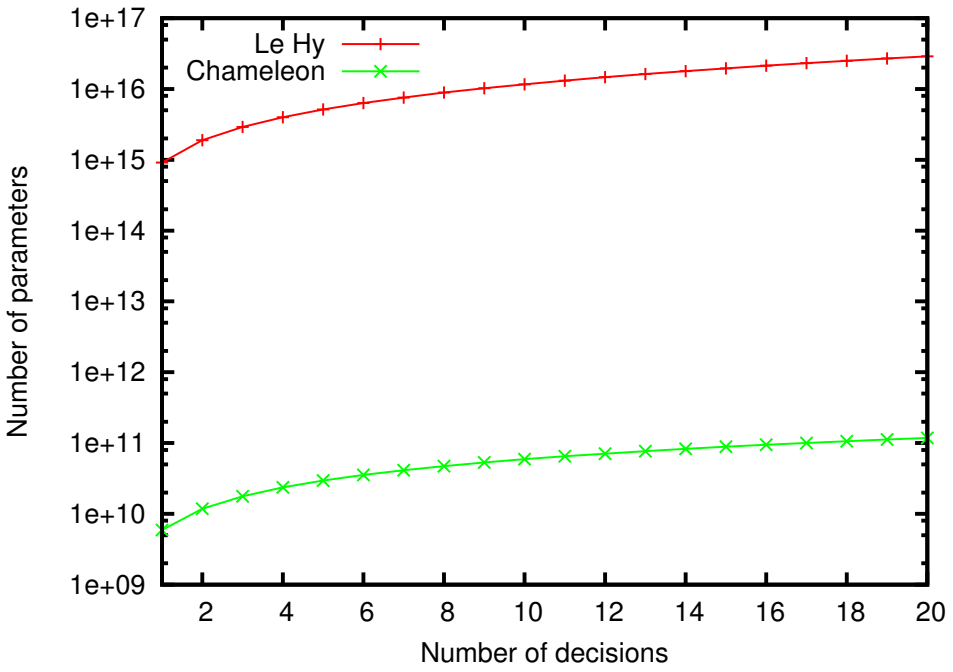


Figure 4.4: *Number of parameters for our application of the model in UT2004. The IP and FEC for Le Hy's model and attention mechanism for Chameleon are not taken into account. The reduction is given compared to Le Hy's model.*

	Number of parameters
Full dependence between random variables	3×10^{18}
Independence of actions	8.6×10^{15}
Independence of actions & semantic refinement	6×10^{10}

Table 4.2: *For each hypothesis, the number of values for the definition of the probability distributions with 10 decisions. The semantic refinement allows a noticeable reduction of the number of parameters. The independence of the actions in introduced in Le Hy's work and the semantic refinement in the previous chapter, section 3.1.*

Conclusion of 4.1

The semantic refinement of the model, clearly defining high and low-level stimuli and the associated actions allows an important reduction of the number of values needed for the definition of the probability distributions. This reduction will make the learning faster because less knowledge is to be learn, the independence between variables being already specified. This should allow the agent to adapt even faster answering to the requirement [B10: Fast Evolution].

4.2 Attention Selection Mechanism

Introduction of 4.2

In order to break the complexity of the model and replace the IP and FEC, we introduced a simple attention selection mechanism. At each time t, the agent focuses on one high-level stimulus and one low-level stimulus. In order to further reduce the complexity, we expressed the distributions to select the stimulus with two functions $\theta(h)$ and $\lambda(l)$. The higher the attention value for a stimulus, the more chances the agent have to focus on it. We will now study the advantages of the attention selection mechanism over the IP and FEC.

Plan of 4.2

In this section we will first explain how to find values for the attention functions which can allow the model to give believable results (section 4.2.1). In the section 4.2.2, we will study the differences between the attention mechanism and the IP and FEC: number of parameters, readability and expressiveness.

4.2.1 Choice of the Values of Attention Functions

As explained in section 3.4.3.3, the values for the attention function are not learnt for now. As a consequence, their values have to be defined manually. Finding values which make the agent behave in a believable manner requires knowledge about the environment. An attention function for an important stimuli must have a high value. As a rule of thumb, stimuli in the centre of the FOV, threats and dangers, extreme values and objects in interaction range should have high values. The farther from these conditions, the lower should be the value.

In our application, specifying the value for all the 572 stimuli is tedious so we defined some rules. For the low-level stimuli we ordered them from the most important to the least:

- *Player* stimuli because all other players are potential threats
- *RayImpact* stimuli make the agent able to avoid collisions
- *Weapon* and *Health* stimuli make the avatar stronger
- *NavigationPoint* stimuli allow the agent to move in the environment

For all of these five stimuli, the closer to the centre of the FOV, the higher the value. For the high-level stimuli, extreme values are important like a high number of enemies, a low health or few ammunition left in the current weapon.

4.2.2 Consequences of Attention Selection

The goal of the attention selection mechanism is the break the complexity of the combination of stimuli without reducing too much the expressiveness of the model. We will study the usefulness of the mechanism comparing to no mechanism and the IP and FEC.

4.2.2.1 Decrease in the Number of Parameters

The goal of the IP is the decrease the complexity of the distributions for the choice of the decision. The hypothesis, which is quite strong, is that stimuli are independent given the decision. The distributions need a total of:

$$|D|^2 + \left(\sum_{i=1}^{N_H} |\mathcal{H}_i| + \sum_{j=1}^{N_L} |\mathcal{L}_j|\right) |D| \tag{4.6}$$

The goal of the FEC is also to decrease the complexity of the distributions but for the choice of the actions. The number of values for the distributions needed for the FEC is:

$$|D| \left(\sum_{i=1}^{N_H} |\mathcal{H}_i| + \sum_{j=1}^{N_L} |\mathcal{L}_j|\right) \sum_{n}^{N_A} |A_n| \tag{4.7}$$

Because of the weaknesses of the IP and FEC, we introduced a simple mechanism for the attention selection. The number of values for each attention function is:

$$\text{for } \theta\text{: } \sum_{i=1}^{N_H} |\mathcal{H}_i| \tag{4.8}$$

$$\text{for } \lambda\text{: } |\mathcal{D}| \sum_{j=1}^{N_L} |\mathcal{L}_i| \tag{4.9}$$

Then Chameleon model needs

$$\left(\sum_{i=1}^{N_H} \mathcal{H}_i\right) |D|^2 \tag{4.10}$$

values for the distributions for the choice of the decision and

$$\left(\sum_{i=0}^{N_H} |\mathcal{H}_i| \sum_{u=1}^{N_R} |\mathcal{R}_i| + \sum_{j=0}^{N_L} |\mathcal{L}_j| \sum_{f=1}^{N_E} |\mathcal{E}_j|\right) |D| \tag{4.11}$$

values for the distributions for the choice of actions.

The difference in the number of parameters for the example of section 3.1.3 is given in Table 4.3 and in figure 4.5. All the modifications we proposed decrease the number of parameters by 36 to 40% compared to Le Hy's model. However, the results for this example cannot be generalised for all the problems. Indeed, the semantic refinement allows an important decrease in the number of parameters whereas the attention selection makes the number of parameters increase. There may be some applications of our model where it needs more parameters than Le Hy's.

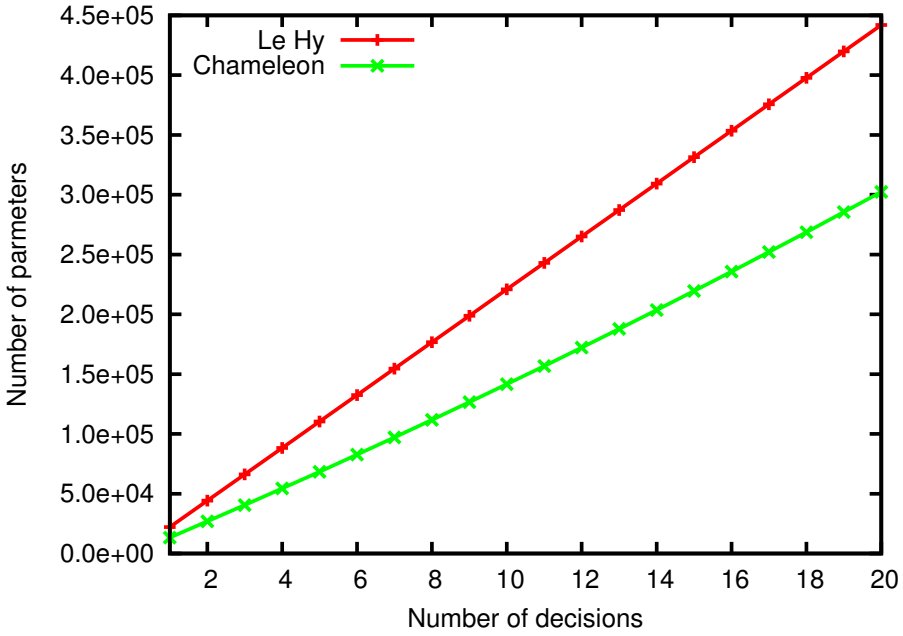


Figure 4.5: *Number of parameters for our application of the model in UT2004. The reduction is given compared to Le Hy's model.*

	No. of parameters
Full dependence between random variables	3×10^{18}
Independence of actions	8.6×10^{15}
Independence of actions & semantic refinement	6×10^{10}
Le Hy: independence of actions & IP & FEC	2.2×10^{5}
Chameleon: indep. of ac. & sem. ref. & attention	1.4×10^{5}

Table 4.3: *For each hypothesis, the number of values for the definition of the probability distributions with 10 decisions. Our model totals 36% less parameters than Le Hy's model. The Inverse Programming (IP) and Fusion by Enhanced Coherence (FEC) are introduced in Le Hy's thesis and the attention selection mechanism is introduced in the previous chapter, section 3.2.*

4.2.2.2 Increase in Expressiveness

As explained previously (see figure 4.6 for a remainder) the FEC cannot express several simple behaviours. In our example, if the attractor are navigation points and the actions forward/backward, the agent will randomly go back and forth, barely moving because the "expected value" is to stay still. There is also an other problem: if a random distribution is 0 for an action, the product is also 0. For our example, one could set the value for going backward when seeing an attractor in front to 0 and *vice versa*. In this case, the FEC cannot be applied because the product would be 0 for all the probabilities.

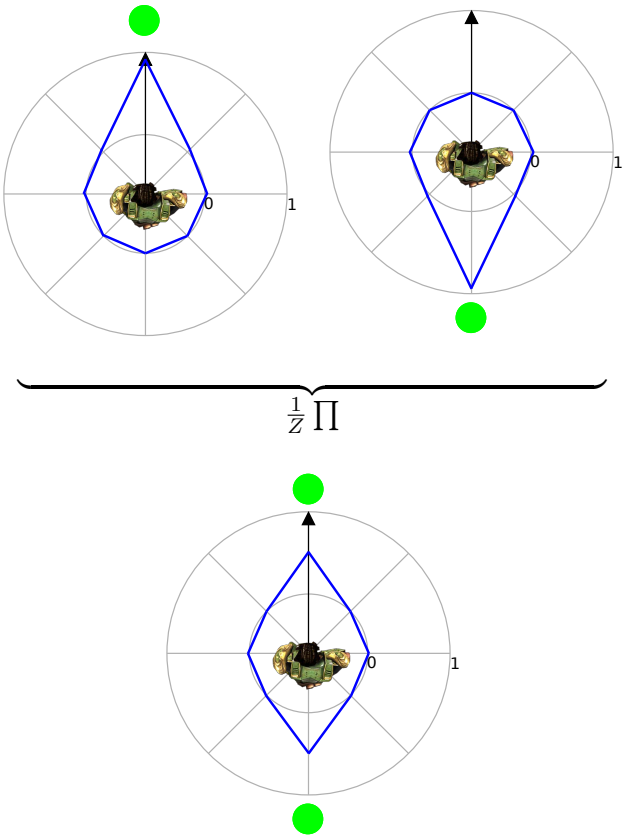


Figure 4.6: *An illustration (same as figure 2.28) of the problem with FEC: on the top, each distribution gives a believable action for a single attractor. On the bottom the FEC does not give a believable action because the agent may constantly switch between the two attractors, oscillating constantly.*

The attention selection mechanism allows the model to solve the problem the FEC suffers (see Figure 4.7) still breaking the complexity with the definition of θ and λ as we saw in the previous section.

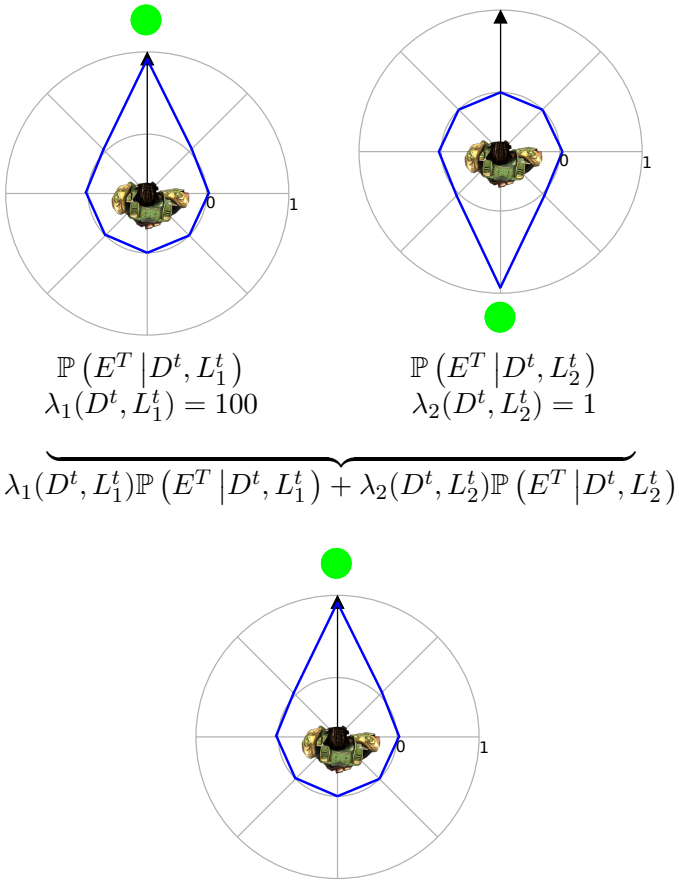


Figure 4.7: *Using the same example as 2.28, the attention selection mechanism produces a better distribution in term of believability: on the top, each distribution (blue line) gives a believable direction to go for a single attractor (green dot). On the bottom the attention gives a believable action because the agent focus on the attractor ahead instead of switching between the two attractors like the FEC would do.*

Conclusion of 4.2

The attention selection mechanism combined with the semantic refinement allows to reduce the number of parameters for the behaviour model and widen the range of behaviours the model can express. Indeed, the model can now express simple but essential behaviour like exploration. This makes our agent more likely to behave in a believable way without the need of a huge number of parameters. We remind that the less the parameters the faster should be the learning.

4.3 Learning The Environment

Introduction of 4.3

For the agent to be able to adapt to unknown environments still being able to move in it in a believable way, we use a modified version of the GNG to learn a representation of the environment. Although this model has already been used in a video game (Thurau et al., 2004), its characteristics and parameters have not been studied for this kind of application. As the results of the model is to be used by the imitation EM, the learning by the GNG must be extremely fast to allow the EM to work on accurate data.

Plan of 4.3

We will first present results and explain how we will measure their characteristics and quality in section 4.3.1. Then we will study the influence of each parameter in section 4.3.2. Finally, as the speed of the learning is very important, we will show some possibilities to increase the speed in section 4.3.4.

4.3.1 Measures and Representation of the Results

4.3.1.1 Application to UT2004

Tracking the position of the teacher in UT2004 is easily done using Pogamut. The position can be given to the GNG for the learning. The difficult part is to find parameters (see section 3.3.2) for the GNG to give a representation with enough nodes for the agent to be able to move in the environment but not too much to avoid overloading the agent with information. We have to choose these parameters in a empirical way because we cannot find them analytically nor use an optimization algorithm. The parameters giving representations close to those usually found in video games are the following:

- $\overrightarrow{attract_1}$: Attraction force applied to the closest node $first$ form the $input$ is 0.03 times the vector $input - first$
- $\overrightarrow{attract_2}$: Attraction force applied to $first$'s neighbours is 0.0006 times the vector $input - first$
- $\grave{Err}$: The error decay for the nodes is 6 Unreal Units (UUs)
- $\overline{Err}$: The maximum error for the nodes is 16000 UUs
- $\overline{Age}$: The maximum age for the edges is 75

In order to compare with other environments, the position in Unreal Tournament is given in UUs (1 meter is roughly equal to 50 UUs).

4.3.1.2 Representation of the Environment

With those parameters we trained two GNGs on two different environments. The first one is a simple environment, called *Training Day*, it is small and flat which is interesting to visualize the data in two dimensions. The second one, called *Mixer*, is much bigger and complex with stairs, elevators and slopes which is interesting to see how the GNG behaves in three dimensions. The results is given in figure 4.8 for the simple environment and in figure 4.9 for the complex environment.

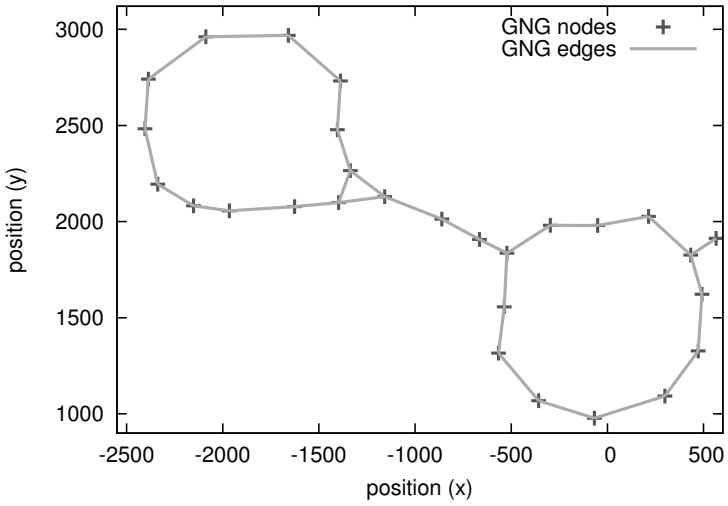


Figure 4.8: *Result of a GNG learned from a player for a simple environment after 30 minutes, top view.*

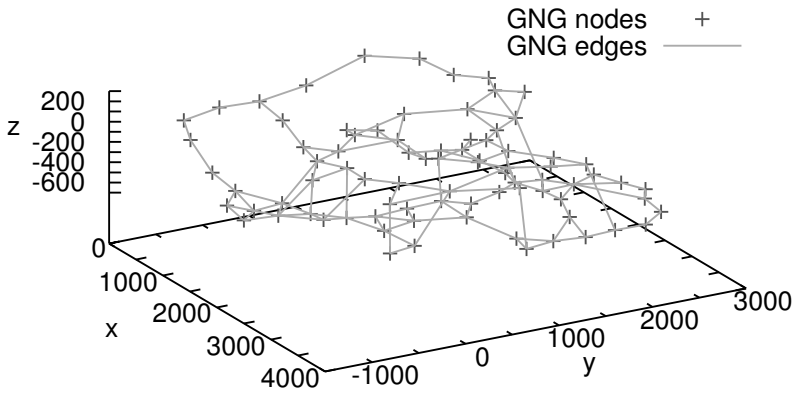


Figure 4.9: *Result of a GNG learned from a player for a complex environment after 1 hour.*

As the complex environment is hard to visualize we focus mainly on the simple one in this section to make the explanation easier to understand. The GNG is however able to represent complex environments as well as simple environments. In the figures, we showed the edges of the GNG to remind its structure but only the nodes are used by the agent in the form of points of interest.

4.3.1.3 Measures of the Time Evolution

In order to study the quality of the learned topology, we first choose to compare the nodes of the GNG with the navigation points placed manually by the environment creators. In the following *node* will always refer to the GNG and *navigation point* to the representation made by the environment creators. We do not want the GNG to fit exactly navigation points but they can help to have a first evaluation of the learned representation. In the game UT2004, we have those navigation points but our goal is that they are not longer necessary for an agent to move in a new environment. The representations learned by the GNG should also allows more believable behaviours as they already provide an information on how players use the environment.

Figure 4.10 shows both the navigation points and the nodes of the GNG for the simple environment. As we can see, the two representations look alike which indicates that the model is very effective in learning the shape of the environment. However, there are zones where the GNG's nodes are more concentrated than the navigation points and other where they are less concentrated. We cannot tell now if it is a good behaviour or not as we should evaluate an agent using this representation to see if it navigates well. Even in the less concentrated zones, the nodes are always close enough to be seen from their neighbours which allows at least node-to-node navigation.

As the qualitative evaluation of the representation is not sufficient, we introduce two measures which principle is borrowed from statistics: *sensitivity* and *specificity*.

Sensitivity Sensitivity measures how much the GNG successfully represents the part of the environment the teacher used which can be seen as true positives. $mindist(a, B)$ is the minimum distance between point a and points in the set B. We computed this measure with the following formula:

$$Sensitivity \propto \frac{1}{\sum_i mindist(NP_i, Nodes)} \tag{4.12}$$

Where NP_i is the i^th navigation point. The higher the value the better the GNG is.

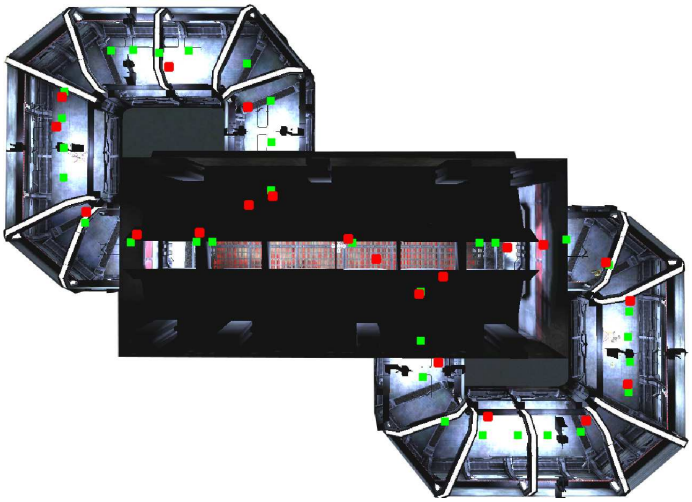


Figure 4.10: *Comparison of nodes learned by the GNG (in red) with the navigation points placed manually by the game developers (in green). The environment viewed from the top is visible in the background.*

Specificity Specificity measures how much the GNG did not represent the part of the environment the teacher did not use which can be seen as true negatives. We computed this measure using the following formula:

$$Specificity \propto \frac{Number\,of\,Nodes}{\sum_i mindist(Node_i, NPs)} \tag{4.13}$$

The higher the value, the better the GNG is.

Number of Nodes In the following, we will also study the number of nodes the GNG has because we do not want the GNG to have either too many or too few nodes.

Figure 4.11 shows this three measures during the learning for the simple environment. For the simple environment, the GNG reached a stable state in approximatively 8 minutes of real-time simulation. For the complex environment, it takes more time, about 13 minutes, but results still continue to improve slowly afterwards. These durations do not depend on the computation power of the machine used to run the algorithm, they only depend on speed of the simulation. In our experiments, we always use the default simulation time of the game, allowing the players to play normally. Those results show that it is possible to have an agent learn during the play.

As the attraction applied to the nodes for each input is constant, the GNG is not converging to a totally stable state. The small variations in the distance in

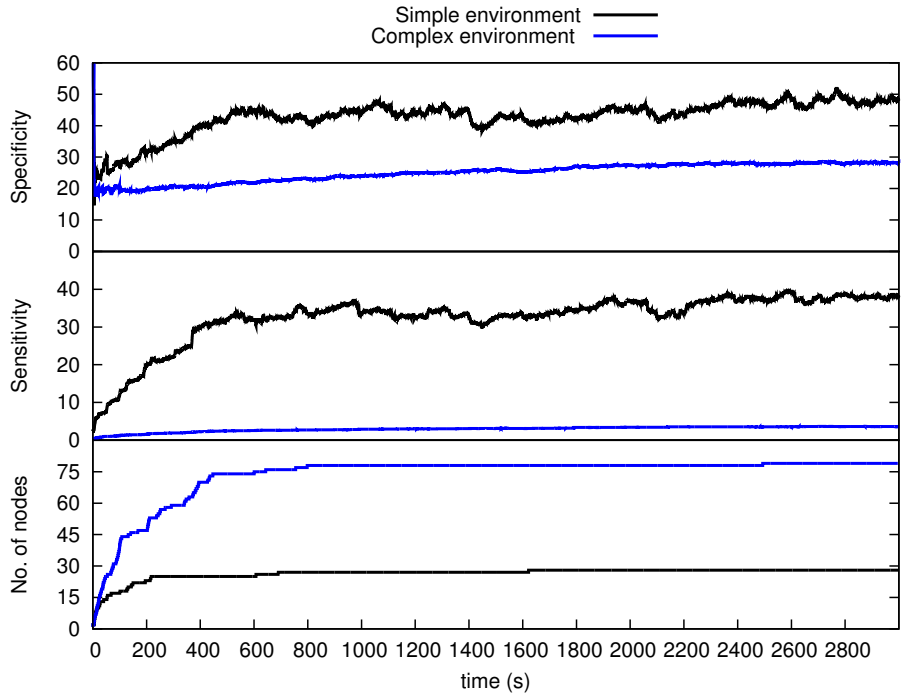


Figure 4.11: *Time evolution of the GNG number of nodes and the cumulated distance between the GNG nodes and the navigation points.*

figure 4.11 shows that the GNG nodes still move. This is a wanted behaviour, allowing the GNG to adapt to a variation in the use of the environment: if the teacher suddenly uses a part of the environment which he/she has not explored yet, the GNG will be able to learn this new part even if the GNG has been learning for a long time.

Even if the GNG is not converging, we do not want it to grow indefinitely. We also want the model to give similar results for similar behaviours. Figure 4.12 shows that even after 10 hours, the number of nodes is not higher than after 30 minutes and even that the solutions represent better the environment. It also shows that two runs of GNGs on similar behaviours give similar results in term of shape and number of nodes. Needless to say that two GNGs learning on the exact same data give the exact same representation.

Models learned on different teachers does not give the same results and the time evolution is also quite different (see figure 4.13). Depending on the behaviour the GNG can reach a stable state much more rapidly. For this reason we will only compare GNG which learned on the exact same data in the following experiments.

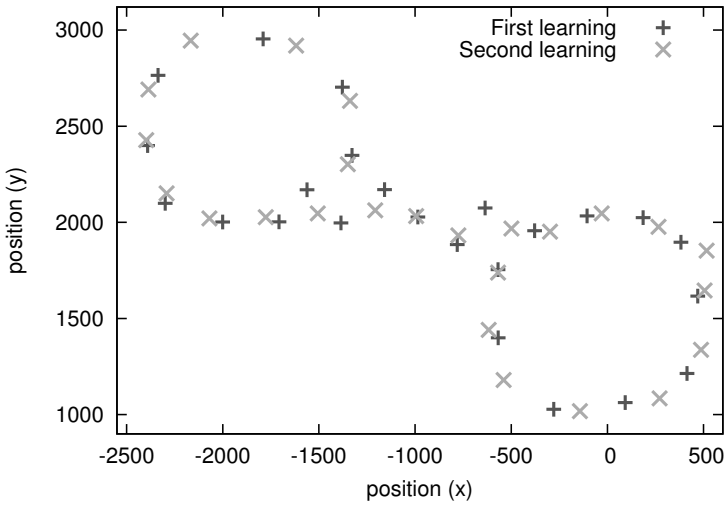


Figure 4.12: *Comparison of two GNGs which learned on the same environment, after a very long training time of 10 hours.*

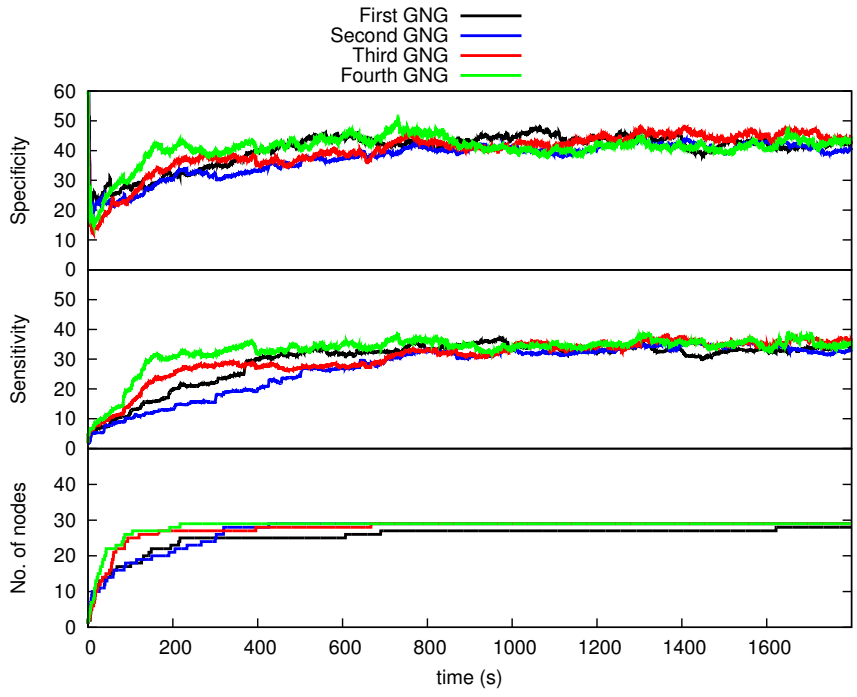


Figure 4.13: *Comparison of four GNGs learned on different players.*

4.3.2 Influence of the Parameters on the Learning

We listed fives parameters for the GNG and gave values for our implementation to give similar results to representations used in UT2004. In order to be able to find those parameters we will now detail the influence on the results for each of the parameters and on the time taken to give an accurate representation. As each of the parameters has an influence on several characteristics of the GNG, we will explain how to choose the parameters after the study of each of them.

4.3.2.1 Attraction of the Winner Node

When an input is given to the GNG, the closest node, the winner, is moved by a certain amount toward this input. We will analyse the impact of the attraction force applied to the winner node with the results in figure 4.14.

Number of Nodes A high force makes the GNG produce less nodes, because as the nodes move closer to the input, their error is lower. Nodes are thus less likely to reach the maximum error causing a new node to be added. Similarly, a low force makes the GNG produce more nodes.

Sensitivity A high attraction force makes the nodes move more. As a result, the GNG is less stable which cause variations in the representation shown by a less stable sensitivity. There is no big difference in the final sensitivity, the bigger sensitivity for the low force must certainly comes from the higher number of nodes.

Specificity As for the sensitivity, a high attraction makes the value fluctuate showing that the GNG is not very stable. However the attraction must be strong enough to attract node which are not representing the environment: a low attraction let too many useless nodes resulting in a low specificity.

Time to Stability A strong attraction makes the model converge more rapidly to a stable state because the nodes are more rapidly spread across the whole environment. This can be seen with the stabilization of the three measures earlier for a high value of the parameter than for the low value.

4.3.2.2 Attraction of the Neighbours of the Winner Node

Once the winner node has been attracted toward the input, the same is done, with a lesser force, to all the neighbours of the winner. The time evolution of the three measures is given in the figure 4.15.

Number of Nodes The force applied of the neighbours has no or very few impact on the number of nodes the GNG creates.

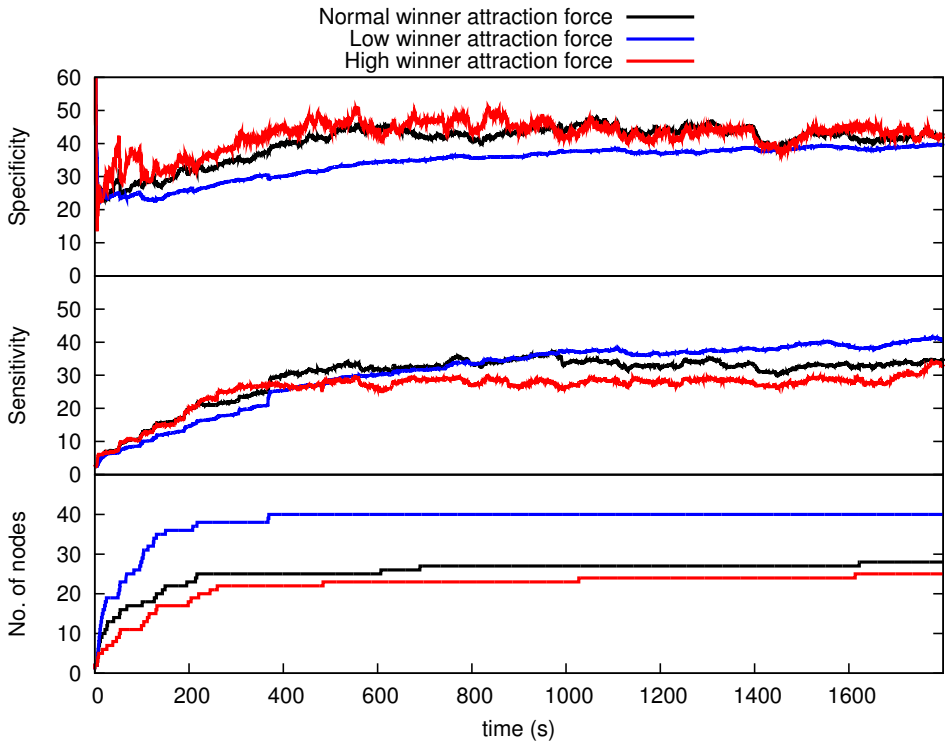


Figure 4.14: *Comparison of the time evolution of GNGs which learned on the same data but have different values for the adjustment of the winner node toward the input. The higher the attraction, the faster the GNG converges, the less nodes the GNG has and the less stable is the representation.*

Sensitivity As for the force applied to the winner, the higher the force applied to the neighbour, the less stable is the GNG. With a low force, the model is more stable and have a better sensitivity because the GNG is more able to generalise.

Specificity However a low attraction force does not allow the GNG to move foreign nodes closer to where the teacher is. As a consequence, the specificity is better with a high force, able to move "false positives" nodes toward a good location.

Time to Stability The attraction force of the neighbours seems to have no influence on the time the GNG takes to reach a stable state. However a high force makes the GNG so unstable that it is difficult to determine when the structure of the GNG finished to evolve.

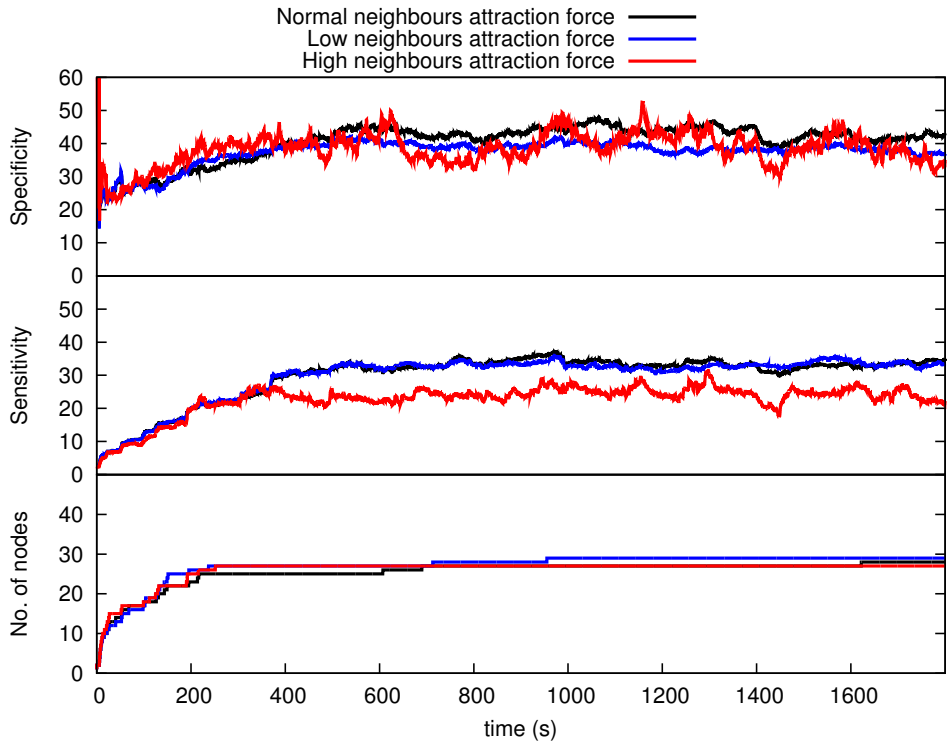


Figure 4.15: *Comparison of the time evolution of GNGs which learned on the same data but have different values for the adjustment of the neighbours of the winner node toward the input. The higher the attraction, the less stable is the representation.*

4.3.2.3 Maximum Error for Nodes

When the nodes has been moved, if the winner node exceeds $\overline{Err}$, a new node is created. We will now study the influence of this amount, the maximum error for a node with the results given in figure 4.16.

Number of Nodes This parameter has a big impact on the number of nodes. The lower the error, the more nodes are created because the more likely they are to exceed the value.

Sensitivity Because a low error makes the GNG produce more nodes, the sensitivity is higher. Indeed, the higher the number of nodes, the more likely a node will be close to a navigation point. Therefore the maximum error seems to have no direct influence on the sensitivity.

Specificity As we already saw, a low error makes the GNG creates nodes more often even if they are not useful for the accuracy of the representation

of the environment. Therefore, a low error makes the GNG specificity higher by lessening the number of useless nodes.

Time to Stability The maximum error does not have a very important impact on the time the GNG needs to reach a stable state. It seems however that a low error makes the GNG converge a bit faster. This may be due to to fast creation of node across the environment.

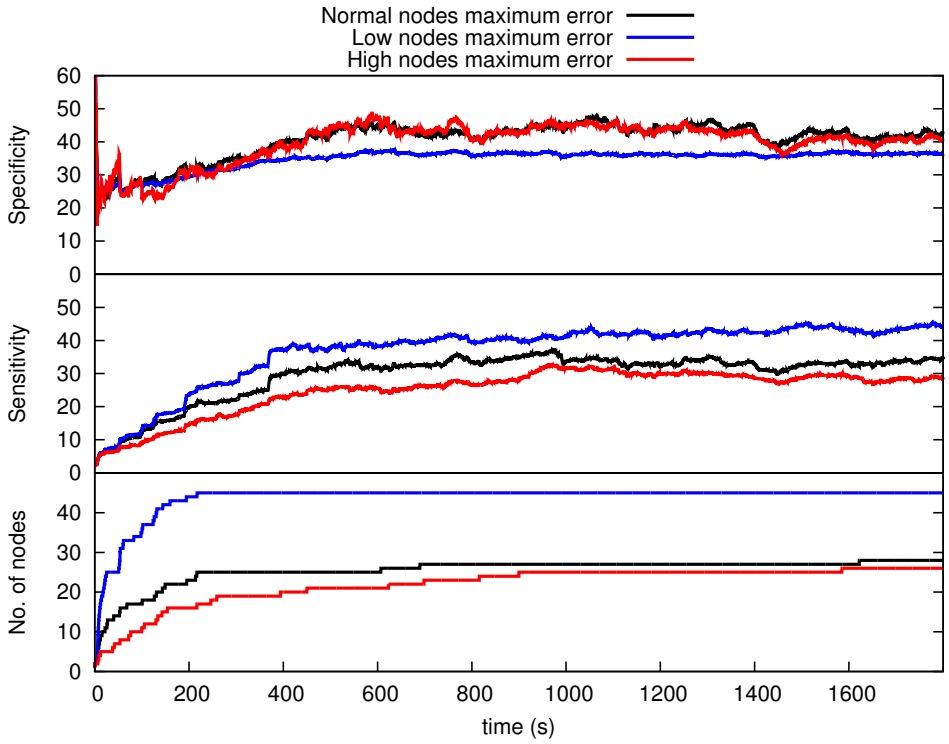


Figure 4.16: *Comparison of the time evolution of GNGs which learned on the same data but have different values for the maximum error admitted for nodes before the creation of another node in the GNG. The higher the maximum error allowed for a node, the less nodes the GNG has.*

4.3.2.4 Maximum Age for Edges

Each time a winner node is selected, each edge connected to this node sees its age incremented by one. If the age exceeds $\overline{Age}$, the edge is deleted. The influence of this parameter will be studied with the figure 4.17.

Number of Nodes The maximum age has no significant influence on the number of nodes.

Sensitivity The maximum age has no significant influence on the sensitivity. Indeed, the small difference comes from the difference in number of nodes.

Specificity Surprisingly, the parameter has no influence on the specificity. We could have expected that edges lasting only a few time step would leave nodes alone where they do not represent the environment correctly. A more detailed study should be done to validate this result.

Time to Stability The maximum age has no influence on time the GNG needs to be stable.

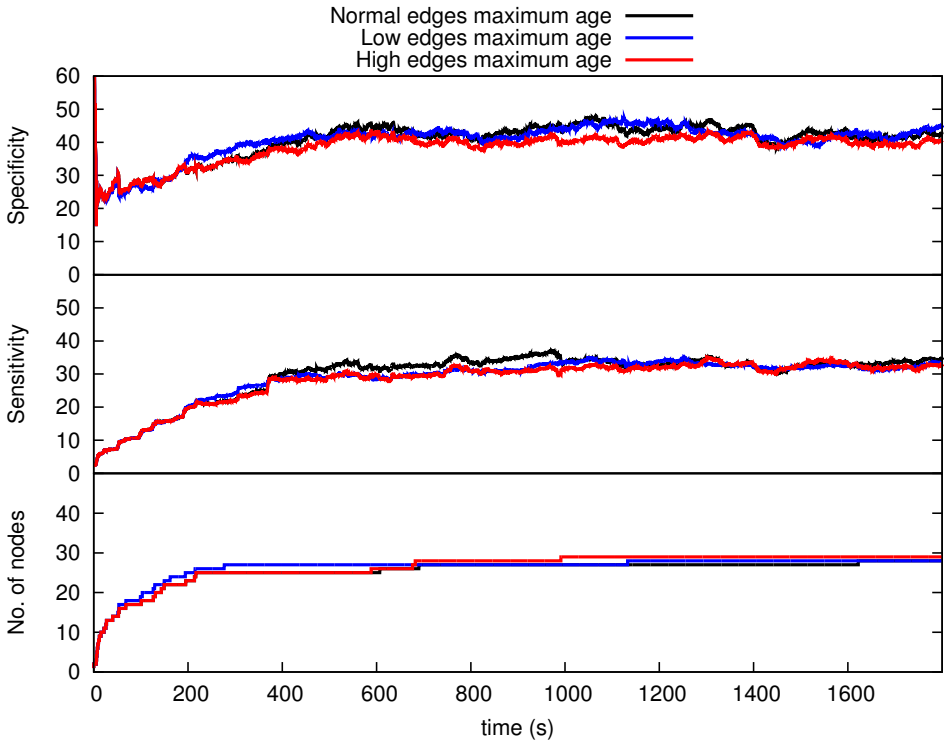


Figure 4.17: *Comparison of the time evolution of GNGs which learned on the same data but have different values for the maximum age admitted for edges before they are deleted. No change on the results has been noted.*

4.3.2.5 Error Decay

Each time step, the GNG reduces the error for each node by $\dot{E}\gamma$. The time evolution of the three measure depending on the value of this parameters is given in figure 4.18.

Number of Nodes A low error decay makes the nodes more likely to reach the $\overline{Err}$, thus makes the GNG create more nodes. Similarly, a higher decay makes the GNG create less nodes.

Sensitivity The difference in the sensitivity comes mainly from the difference in the number of nodes, thus the parameter does not affect directly the sensitivity.

Specificity A high decay increases the specificity of the model. We are not sure about the reasons of this behaviour. It may be because the model reaches faster its maximum number of nodes, thus the GNG can improve the representation without the disturbance of new nodes.

Time to Stability The GNG reaches a stable state faster with a high decay because the error shared between all the nodes is rapidly compensated by the decay. Similarly, a low decay makes the GNG very slow to converge because new nodes are added long after the beginning of the learning.

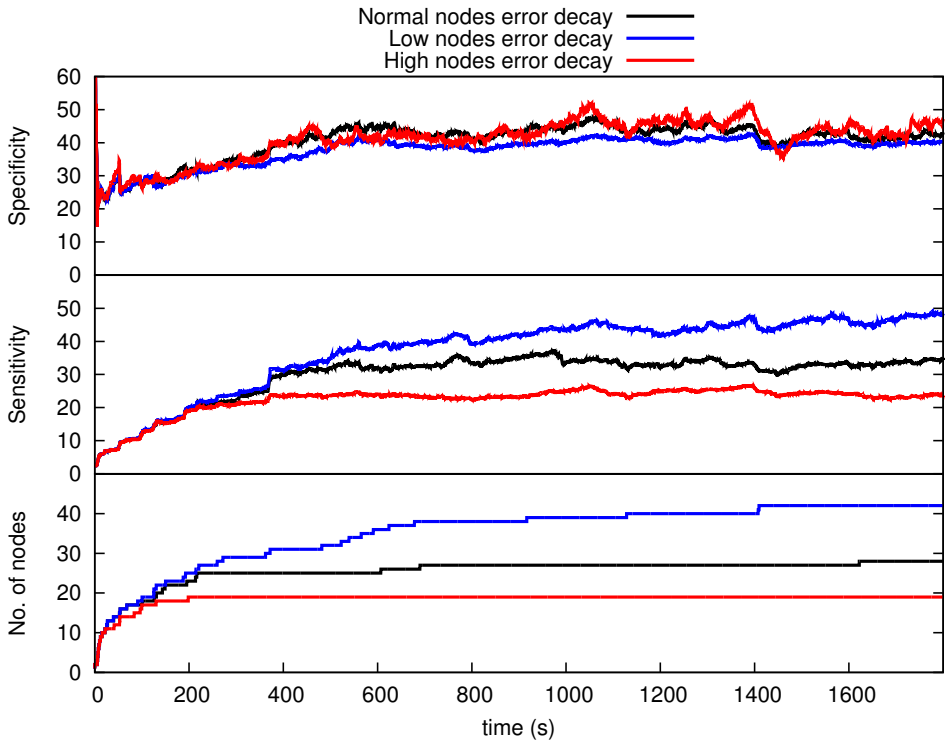


Figure 4.18: *Comparison of the time evolution of GNGs which learned on the same data but have different values for the decay of the error of the nodes. The higher the error decay, the less nodes the GNG has and the higher the specificity.*

4.3.3 How to Choose the Parameters

The parameter $\overline{Age}$ can be chosen without problem because it does not influence the results. A value of 0 should be avoided as it would impede the attraction of neighbours which has an impact on the results.

A high $\grave{E}\searrow r$ can be then used to reduce the time of converge of the model. However this value is linked to the $\overline{Err}$, so if the decay is very high, the maximum error must be too.

A rather high force should be applied to the winner, keeping in mind that a too high value will make the GNG less stable. If the teacher often discovers new areas, the force should be high to allow the GNG to adapt more quickly.

In a similar manner, the force applied to the neighbours should be high enough to avoid having useless or even harmful nodes for the believability. However a high force will also makes the GNG less stable.

Finally, the $\overline{Err}$ can be chosen according to the value of $\grave{E}\searrow r$ so that the representation gives enough nodes for the agent to be able to move in the environment but not too much to avoid overloading it with information. The notion of "overloading" depends mainly on the behaviour model: if it is capable of handling much information, $\overline{Err}$ can be set to a low value.

4.3.4 Increasing the Speed of the Learning of the Representation

As explained at the beginning of this section, the speed of the learning of the representation of the environment is very important. If the agent has not a correct representation of the environment, he cannot move in it and worse, he cannot learn any behaviour. Indeed, the learning algorithm cannot associate the actions to the stimuli if the stimuli are not yet defined. We will see how to increase the speed of the learning without compromising the quality of the representation.

4.3.4.1 Learning on Several Teachers

The GNG can handle inputs from several teachers. Figure 4.19 shows our three measures for GNGs trained on one, two, three and four teachers. The learning multiple teachers is a bit faster. Moreover, learning with multiple teachers gives GNGs with better results in sensitivity and specificity, the number of nodes been the same. As a consequence, the GNG should be learned on the most players available to have better results. Moreover, the number of teachers should be higher for vast environments to speed up the learning.

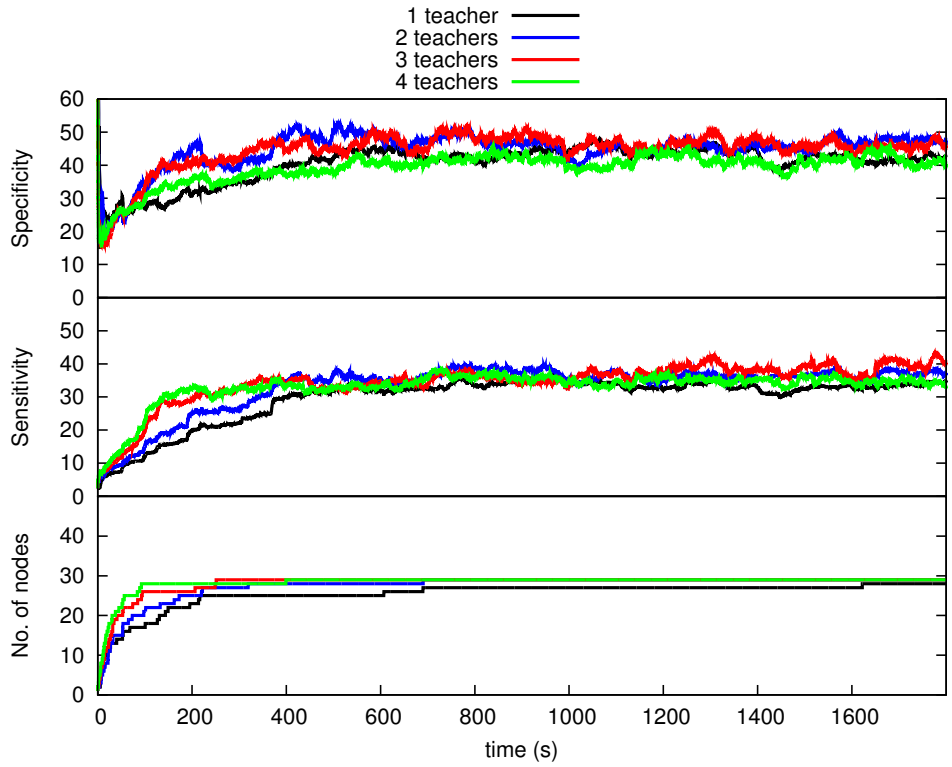


Figure 4.19: *Comparison of the time evolution of GNGs which learned on simultaneously 1, 2, 3 and 4 different teachers. The more the teachers, the faster the learning but also the more stable is the representation.*

4.3.4.2 Input Frequency

The last evaluation assesses the impact of the frequency at which the demonstrator's position is given to the GNG. For the previous experiments, the frequency was set at 10Hz. Figure 4.20 shows the differences for 1, 5, 10 and 20 Hz (Pogamut does not allow higher than 20Hz). Results indicate that the higher the frequency, the faster the GNG stabilizes and the better the results in term of sensitivity and specificity are. With high frequencies the number of nodes is a bit higher but the main problem is the stability of the representation. The sensitivity and specificity vary a lot at 20Hz. A variable frequency could be used to speed up the learning at the beginning and to avoid instability when most of the learning is done.

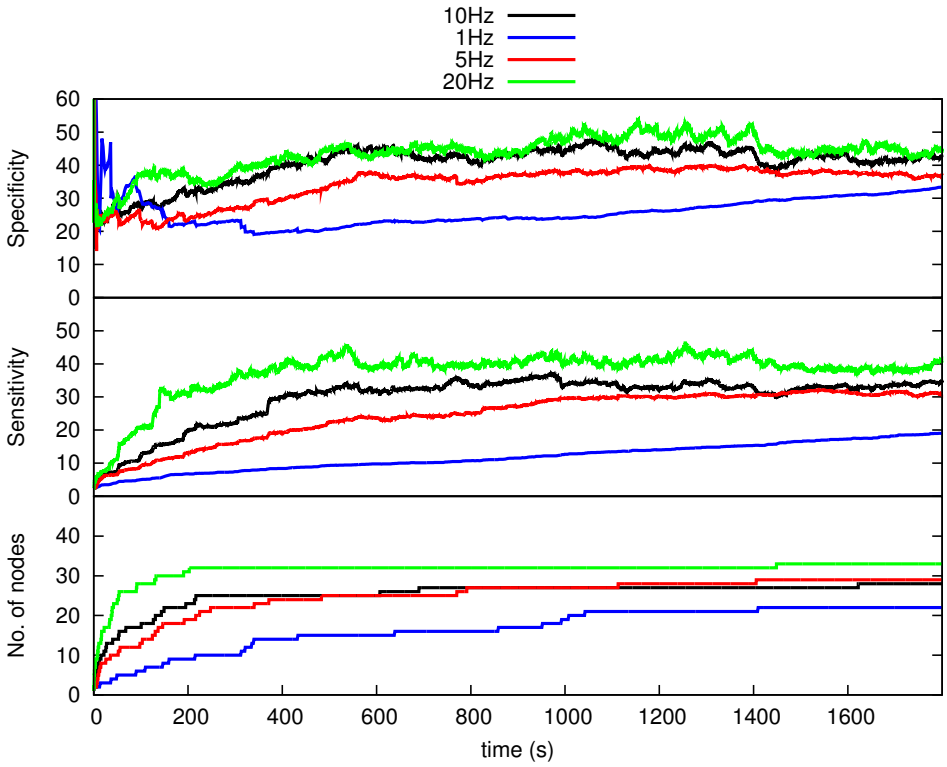


Figure 4.20: *Comparison of the time evolution of GNGs which learned at 1, 5, 10 and 20Hz. The higher the frequency, the faster the learning but the less stable is the representation.*

Conclusion of 4.3

The GNG proves to be very efficient at learning the topology of an environment by imitation. The learning is quite fast, even for complex environments. Although the model does not converge toward a unique and totally stable solution, the representations are similar in shape and accuracy. The advantage of this constant evolution is that the model can adapt quickly to changes in the use of the environment. We detailed the influence of each parameter and explained how to find good values for them.

4.4 Learning the Parameters of the Model with an EM Algorithm

Introduction of 4.4

In the previous chapter we have redesigned the learning algorithm from scratch to allow the agent to learn all the parameters of the model by imitation in a much more detailed manner. The goal is to make the agent able to evolve [B9: Evolution], adapting to new games and new conditions with minimum work from the character designers. We also want the algorithm to be fast for the player to notice the evolution [B10: Fast Evolution].

Plan of 4.4

In this section, we will first study the influence of the data given to the EM: the observation sequences and the model parameters (section 4.4.1). Then we will study the convergence of a learning on one sequence and multiple sequences (section 4.4.2). Finally we will try to evaluate the resulting behaviours with different objective and subjective measures (section 4.4.3).

4.4.1 Impact of the EM and Parameters on the Results

Observation Sequences The quality of the results given by the learning algorithm depends on the data given to it. The observations must give accurate information on the interactions between the teacher and the environment. For the model to reflect the behaviour of the agent, the information given to the learning algorithm is the one the agent would have if it was into the shoes of the teacher. We will see however in section 4.4.1.1 that giving the raw observations may not give very good results because the reaction time of the teacher must be taken into account.

Model Parameters Initialization The initial model parameters have also an important impact on the quality of the results. As we did not manage to make the algorithm learn the attention function, we specified them using our knowledge of the game. For the distributions, we used a random initialisation, which is a classic method when no knowledge is given *a priori* to the model. The last parameter is the number of decisions in the model. We will study the impact of the number of decisions in section 4.4.1.2.

Likelihood and Time Measures In the following, the log-likelihood of the model parameter will be studied. It corresponds to the function $Q(\Phi, \Phi_n)$ (see equation 3.24). The higher the value, the more likely the model and its parameters will generate the sequence of observations. We will also give some measures of the time taken for the algorithm to converge. The experiments were run on a machine with an Intel Xeon E5630 2.53 GHz processor and 12GB

memory. In our implementation, an EM algorithm uses one of the four cores the processor has, so the time given is for one core. As a consequence, four EM can run in parallel, one one each core. Thus, if the time of a learning is four time the time during which the sequence was recorded, the whole learning algorithm can handle all the observations without delay. In other word the algorithm can update the parameters with all the data without storing them for latter use.

4.4.1.1 Impact of the Teacher's Reaction Time

When the teacher is observed by the learning algorithm, a snapshot of the values of the stimuli and the actions is taken at each time step t. However, the teacher actions at time t may not reflect a reaction to the stimuli at time t. We must then take into account the reaction time of the teacher, allowing our model to find the relation between stimuli and actions which are actually related. This problem is linked to the requirement [B2: Reaction time]: the model and the learning algorithm must approximate the teacher's reaction time to able to provide believable behaviours. Figure 4.21 shows how the reaction time affects the use of the observation sequences.

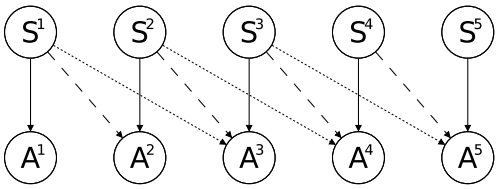


Figure 4.21: *Example of the meaning of reaction time for a simple sequence of 5 observations. The arrows give the association of stimuli/actions given to the learning algorithm to update the model parameters. Considering the observation are taken every 100ms, the plain arrows represent a reaction time of 0ms, the long dashed arrows represent a reaction time of 100ms and the short dashed arrow a reaction time of 200ms. Note that a small part of information is lost when the reaction time is increased.*

The reaction time may not be the same between individuals and may also vary for one individual over the time. However, we find that the simplest solution, using a constant reaction time for all the teachers, give the best results. Other methods were tried, like associating the more likely action to the stimuli according to the current model parameters, or aligning variations in both stimuli and actions but they did not make the learning algorithm converges toward parameters more likely to generate the observations. The choice of the reaction time for a game has to be done once and for all using a graph like figure 4.22.

According to figure 4.22, the reaction time of a player is very variable. The reaction time of 300ms gives the maximum likelihood, but the difference is not very important with the likelihood found for reaction times between 400ms and 800ms. For now, we will use a reaction time of 300ms in the following experiments, but a variable time of reaction could improve the results. Note that a reaction time of 100ms to 200ms is extremely fast. However, the fact the teacher is able to anticipate events may be interpreted by the learning algorithm as very fast reaction times.

In order to show that the results of the learning are really impacted by the reaction time, we studied the variations of the log-likelihood depending on the reaction time for a UT2004 agent teacher (all the other experiments are done with human teachers). Figure 4.22 show that best log-likelihood are achieved for reaction times between 0 and 200 ms. These results are coherent with the fact that UT2004 agents do not model the reaction time. It also shows that the players' behaviours are much more complicated to learn, their reaction time being much more variable.

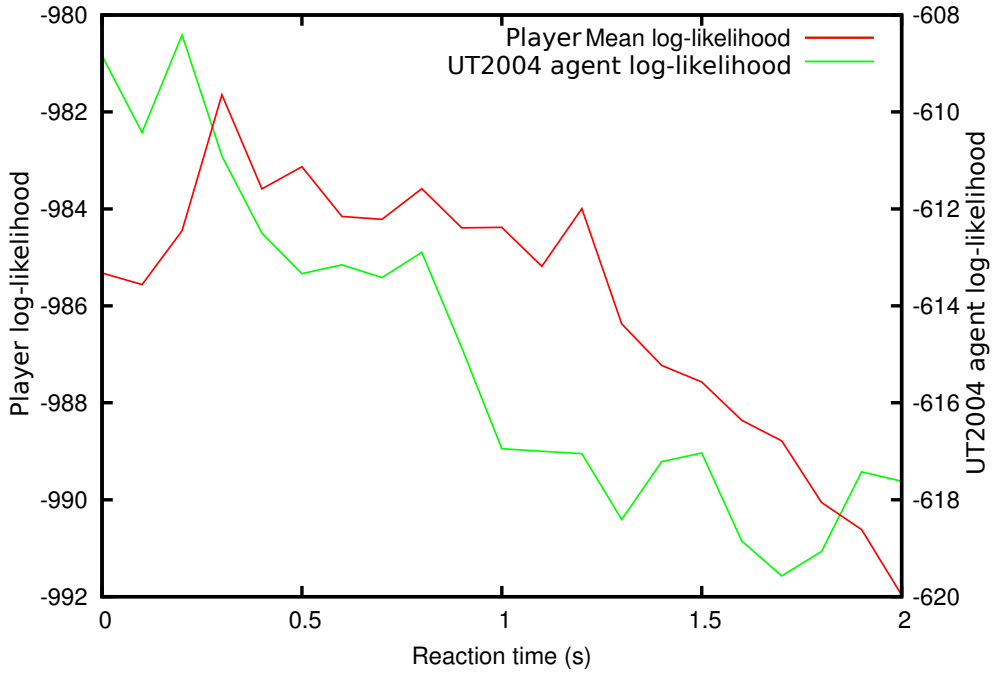


Figure 4.22: *Mean log-likelihood of the final result after learning on 105 different sequences of observations of the behaviour of a player and 100 different sequences of observations for a UT2004 agent. The reaction time varies from 0 second to 2 seconds and the model has 10 decisions.*

4.4.1.2 Impact of the Number of Decisions

The mechanism of decisions allow the agent to produce logical sequences of actions, to make actions according to its needs and to simulate a short time memory. As expected, the figure 4.23 shows that the more the decisions, the better the model can express the observed behaviours. Indeed, the model is more complex and can make the agent behave in a more subtle way. However, as the number of parameters increase with the number of decisions (see section 4.2.2.1), the time needed for the algorithm to converge is longer (see figure 4.23). Too many decisions should be avoided to make the learning fast and fulfil the requirement [B10: Fast Evolution].

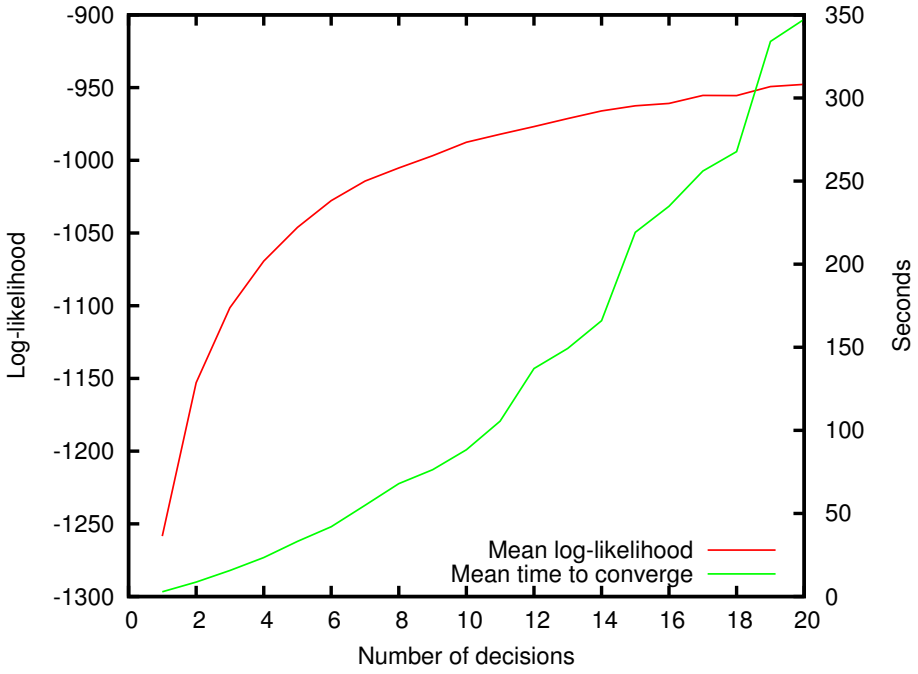


Figure 4.23: *Mean log-likelihood of the final result and time to converge for 105 different sequences of observations. The number of decisions varies from 1 to 20 with a fixed reaction time of 300ms.*

According to the results, increasing the number of decisions over 10 does not improve the results in a significant way. For 10 decisions, the algorithm takes an average 88 seconds to converge for a sequence. The mean duration of the sequences is 23 seconds. As explained in the beginning of section 4.4.1, the convergence time must be lower than $4 \times 23 = 92$ seconds for the learning to be in "real time" on the computer we used for the tests. Therefore, we can consider that 10 decisions is a good choice for the rest of the experiments. We remind the reader that agents originally coded in the UT2004 game use approximatively 10 main states.

4.4.2 Characteristics of the EM

According to the two previous studies, we will use 10 decisions and a reaction time of 300ms for the following experiments. For now, we have only studied the final results and we considered the total likelihood. We will now see how the likelihood evolve for one sequence of observations and for several sequences. We will also look at the likelihood of each distribution to see how they evolve during the learning.

4.4.2.1 Evolution of the Likelihoods

The function Q, is the function which is optimized by the EM algorithm. The higher it is, the higher is the log-likelihood. For this reason, in the manuscript the term log-likelihood is often used in lieu of Q. The function Q is composed of several parts (equations 3.31, 3.32, 3.33, 3.34, 3.35 and 3.36), some of which are independently optimized. We will use the term total log-likelihood for Q and partial log-likelihood for the cited equations. In the following we will study the evolution of the different log-likelihoods during the learning.

The following graphs represent the log-likelihood (total or partial) for 20 different sequences. In order to be able to compare the results, we truncated the sequences to 10 seconds. We also initialised the learning algorithm with the same parameters which are chosen randomly at the beginning of the experiment.

Total Likelihood Figure 4.24 shows the value of the total log-likelihood for each iteration of the EM. The first iterations make the value increase sharply, after 10 iterations the increase becomes very slow, stabilising between -500 and -300. As the value is a log, this likelihood is very small. However, as we will see in the following, it corresponds to parameters which fit well to the data. We can also see that some learnings finish faster, in about 30 iterations, while others take around 100 iterations.

Markovian Distributions Likelihood Figure 4.25 shows the evolution of the partial log-likelihood for equations 3.35 and 3.36. The likelihood of m^1, the distribution giving the initial decision, converges to values very close to 0 meaning the distribution fits perfectly to the data. It usually takes no more than 15 iteration to converge but in some cases it can take much longer. For m, the distribution giving the decision for the time steps after the first, the convergence is usually slower, the value stabilizing in 25 iterations. Depending on the sequence of observations, the final result is variable, from -40 to -20. Globally, the Markovian distributions converge to solutions with a quite high

likelihood but some *a priori* knowledge could reduce the time to converge and improve the final results.

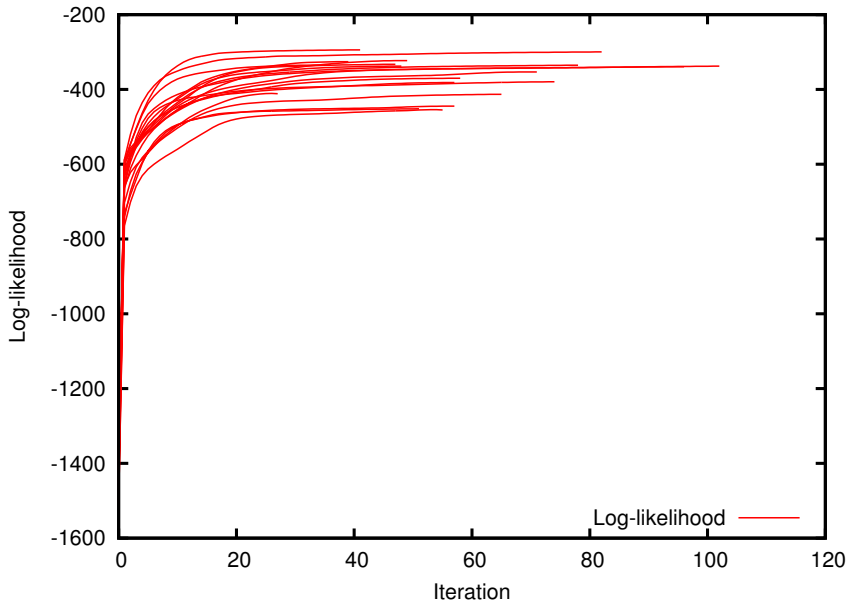


Figure 4.24: *Time evolution of the total log-likelihood for 20 learnings on sequences of same length. Each learning starts the same initial distributions.*

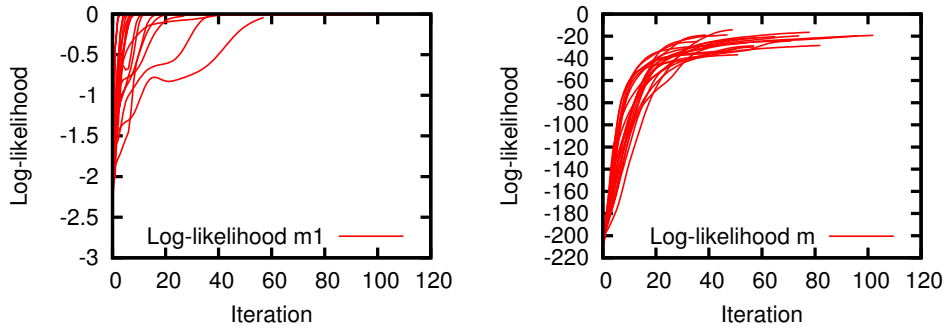


Figure 4.25: *Time evolution of the partial log-likelihood for 20 learnings on sequences of same length for m^1 (left) and m (right) distributions. Each learning starts the same initial distributions.*

Output Distributions Likelihood Figure 4.26 shows the evolution of the partial log-likelihood for equations 3.31 and 3.32. These distributions control both the reflexive and external actions of the agent. Both distributions converge very fast, in less than 10 iterations, toward a good solution with

a log-likelihood between -30 and 0. Even if the result is not "perfect" (a log-likelihood of 0), the results are very satisfying.

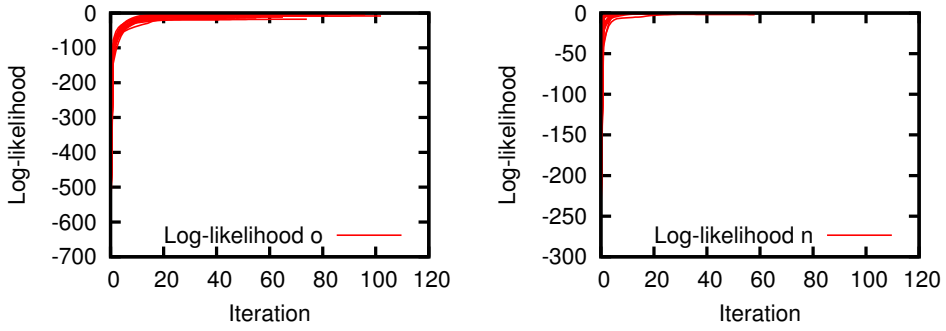


Figure 4.26: *Time evolution of the partial log-likelihood for 20 learnings on sequences of same length for o (left) and n (right). Each learning starts the same initial distributions.*

Attention Functions Likelihood Figure 4.27 shows the evolution of the partial log-likelihood for equations 3.33 and 3.34. These are the partial log-likelihood of the attention functions. As they are not optimized during the algorithm, their values do not converge toward a good solution. Due to the optimization of the other distributions, the log-likelihood of the attention function increase a bit in the first iterations. They are mainly responsible for the low total log-likelihood, their value being comprised between -230 and -60 (we remind the reader that the total log-likelihood converges to values between -500 and -300).

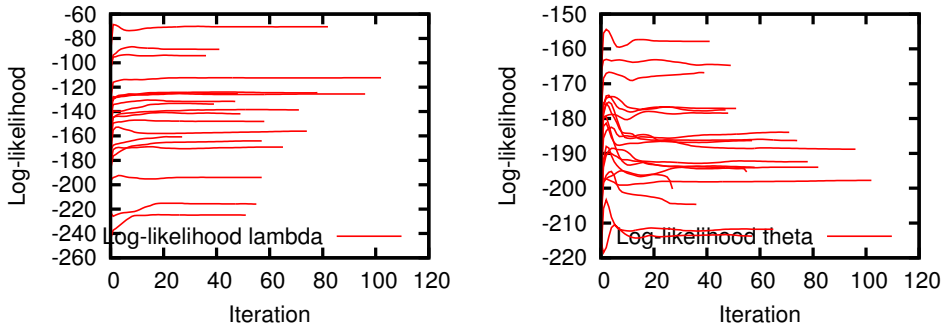


Figure 4.27: *Time evolution of the total log-likelihood for 20 learnings on sequences of same length for λ (left) and θ (right). The value of the attention function are not optimized during the algorithm and we specified their values.*

Most distributions converge well to a log-likelihood between -45 and 0. The simplest distribution, m^1 converges to log-likelihoods very close to 0 because it has few values and the learning algorithm can easily choose an arbitrary decision to begin with. The n and o distributions see their likelihoods increase strongly. Even if they can represent a high number of values, the algorithm behaves well. The Markovian distribution m does not increase as well as the others. It may comes from the fact that decisions are hidden variables and thus hard to estimate. Finally, the non-optimization of the attention functions hinder the efficiency of the algorithm. However, as we specified them knowing how the agent must behave, they can represent well the general behaviour of the teacher but not the particular sequence of observation.

4.4.2.2 Effect of the Merging of the Parameters

The merging of the parameters allows the agent to gather all the results from the EMs into a global set of parameters (see section 3.4.4.3). These global parameters will be used in the model controlling the agent. The idea is to learn while the teacher is still playing, treating short sequences instead of waiting the end of the demonstration to learn the whole behaviour.

The problem is that each learning by the EM find the optimal parameters for one sequence of observation. By using only limited information, the parameters suffer from what is called *over-learning*: the parameters fit too well the data and do not generalise to similar situations. By mixing the learned parameters, we can make the agent generalise over more complete knowledge of the teacher's behaviour.

In order to measure if the merging procedure is indeed allowing the agent to generalise we use a standard test for categorization models. 90% of the sequences of observations are used for the learning while the other 10% is used to test if the Chameleon model them well. Figure 4.28 shows three measures:

- Mean initial likelihood: the mean log-likelihood of the global parameters to generate all the test sequences.
- Mean final likelihood: the mean log-likelihood of the parameters learned on the test sequences using global parameters as initialisation.
- Mean learning time: the mean time needed for the learning on the test sequences.

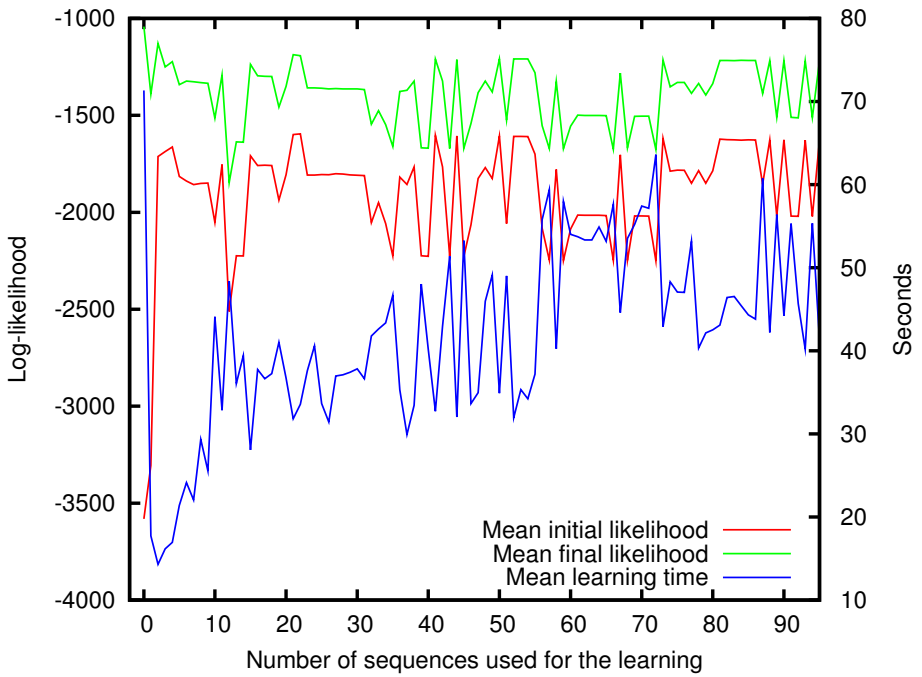


Figure 4.28: *Effect of the merging of multiple results on the log-likelihood and time to learn for test sequences. The mean log-likelihood for the test sequences is given before and after the learning using the global parameters produced by the learning on n sequences. The mean learning time is also given for the test sequences. 95 sequences were used for the learning and 10 for the testing.*

Figure 4.28 shows that the merging of all the parameters has an impact on all the three measures. The mean initial log-likelihood increases from -3500 to around -2000 in only two learnings and then stays between -2750 and -2250. This means the learning and the merging of the parameters allows the global parameters to represent better the behaviour of the teacher even for observations which were not used for the learning. The initial log-likelihood is of course lower than the final log-likelihood which is between -2250 and -1000. The final log-likelihood decreases a from -1000 to around -1400, meaning that the global parameters are not as good as random parameters for the initialisation of the EM. Whether it is a good or a bad behaviour is difficult to assess now but it indicates that the global parameters may be improved to make the EM reach better local maximum. Finally, the learning time is reduced from 70 seconds to around 50 seconds which allows an even faster learning, making the learning algorithm able to cope with lots of sequences, from several teachers for instance.

4.4.2.3 Sequence of Decisions

With the estimators of the EM, it is possible to have for each time step the probability for the model to be in certain configurations of hidden states. It may be useful to study the sequence of decisions the teacher followed in order to study his/her behaviour. As the decisions do not correspond to a defined behaviour before the learning, it is not possible to understand the sequence of decisions *a priori*. A solution is to use an annotated sequence of observations to associate decisions with events in the environment. Table 4.4 shows the sequence of the more likely decisions and some stimuli and actions.

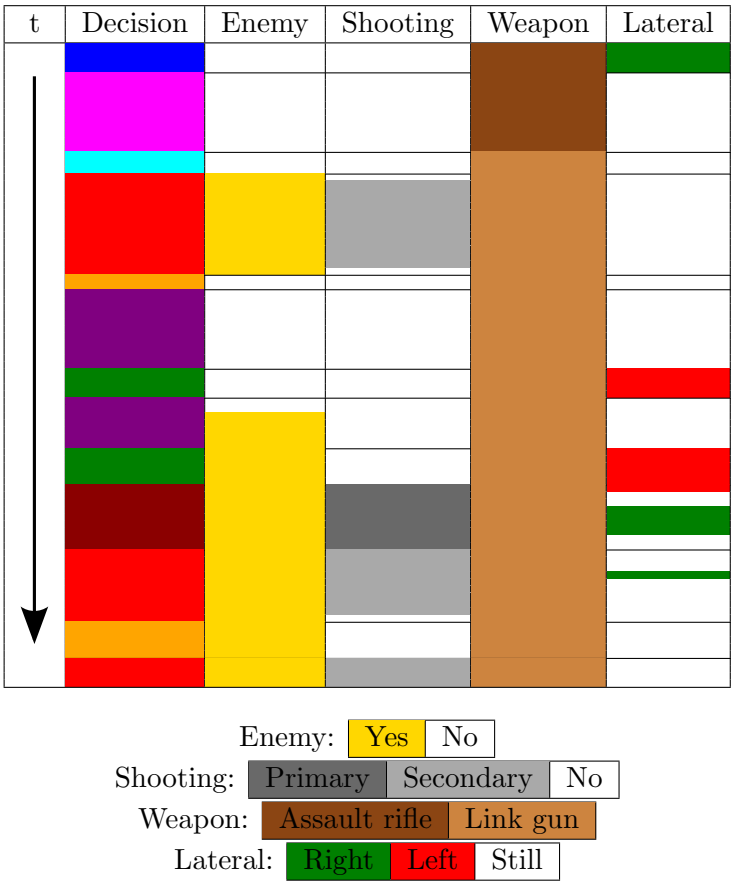


Table 4.4: *Sequence of decisions and values of some stimuli and actions for a sequence of observations. The decisions are those the more likely to be picked (γ_d^t is the higher) according to parameters learned on 100 sequences of observations.*

The teacher begins with the blue decision which seems to correspond to a exploration with lateral movement. Then it switches to the magenta decision where it explores without lateral movement. During this decision the teacher

finds a new and more powerful weapon. After some time steps, it switches to the cyan decisions, equipping the new weapon. Once an enemy appears, the teacher switches to the red decision, attack, and uses its weapon in secondary mode. When the enemy is defeated (or disappear), the teacher changes to the orange decision which seems to be a careful state. Then the teacher begins again its exploration, maybe hunt, with the purple decision (no lateral movements) and green decision (lateral movement). During its exploration, it seems to miss the enemy, actually seen it but not reacting to its presence. When it finally notices the enemy, the teacher attacks it with the primary mode of the weapon (dark red decision) and secondary mode (red decision). During some time steps, the teacher seems to try to aim at the enemy without success (careful orange decision) and then goes back to the red attack decision. Finally, the sequence ends, meaning the teacher was defeated by the enemy. This example shows that it is possible to tag the decisions manually and then be able to describe automatically the behaviour of a teacher.

With the previous studies, we observed that the implementation of the EM is converging toward a local maximum which is conform to the theory. The behaviour of the merging method is quite satisfying, converging rapidly toward a stable solution. However, this solution may not be the best one because it impede the learning to reach the best local maxima for each sequence. As the method allows an increase in the speed of learning, the whole learning algorithm can handle sequences coming from more than one teacher without delay. Finally, the estimators from the EM allows the identification of some teacher's behaviours. It could be used to guess the other players' intentions and could allow the agent to anticipate [B7: Planning].

4.4.3 Resulting Behaviours

The previous studies gave an overview of the characteristics of the learning algorithm, the following ones aim at studying the behaviours produced by the model with the learned parameters. The final goal of the four propositions (semantic refinement, attention selection, learning of the environment and learning of the behaviour) is to make the agent produce believable behaviours.

In the following experiments, the model learned on one unique teacher using 10 decisions. It also considered that the teacher had a constant reaction time of 300ms. The results given are after learning on the teacher during 40 minutes. The teacher played against a UT2004 agent in the environment named in the game *Training Day*.

4.4.3.1 Study of the Distributions

Before analysing the whole behaviour we can study the distributions of actions to see if they really look like the ones in figure 2.15 and in (Le Hy, 2007, page 47, in French). We will focus on the o distributions, giving the external actions according to a low-level stimulus and the current decision. the n distributions is much more simple to learn, therefore we will not study them.

In the following, we will show distributions for the actions:

- *Yaw*, in red, gives the probability for the agent to turn left and right by a certain amount of degrees. The graph can be seen as a view from the top of the agent which is facing 0°: 0° means the agent does not turn, angles less than 180° mean the agent turn right and angles more than 180° mean the agent turn left.
- *Run and lateral*, in blue, gives the probability for the agent to move forward, backward and slide left and right. As for the *yaw*, the graph can be seen as a top view of the agent: 0° is forward, 180° backward, 90° right and 270° left. The sum of the distribution is comprised between 0 and 2 because the agent may choose not to move and because it represents two distributions: lateral and forward/backward.
- *Pitch*, in green, gives the probability for the agent to move its head up and down. Unlike the other distribution, the graph can be seen as a side view of the agent which is facing 90°: 90° means the agent looks at the horizon, 0° means the agent looks straight up and 180° straight down.

For the same decision, the distributions shown in figure 4.29 confirms that the agent does not react in the same way for different positions of a same object. In the example, if a weapon is at the left, close and at the same height as the agent (left figure), the agent will go forward, move laterally left and not turn. If the weapon is at the right and at the same height (right figure), the agent will probably not turn go forward and move laterally right. In the two cases, the agent will look in the direction of the horizon which is the direction of the weapon in term of pitch. In the two cases, the agent will surely pick the weapon.

For the same stimulus but different decisions, the agent may act differently. The figure 4.30 shows the distributions for an enemy player on the right of the agent, moving to the right, at the same height and at an average distance for two different decisions. In the first decision (left figure), the agent moves forward and turn right in order to aim at the enemy and reduce the distance to the player. In the second decision, the agent turns also right but may move

forward or backward and move laterally to the left in order to aim at the enemy but keep the same distance to the player. In the two cases, the agent looks at the direction of the horizon as it is the direction of the enemy.

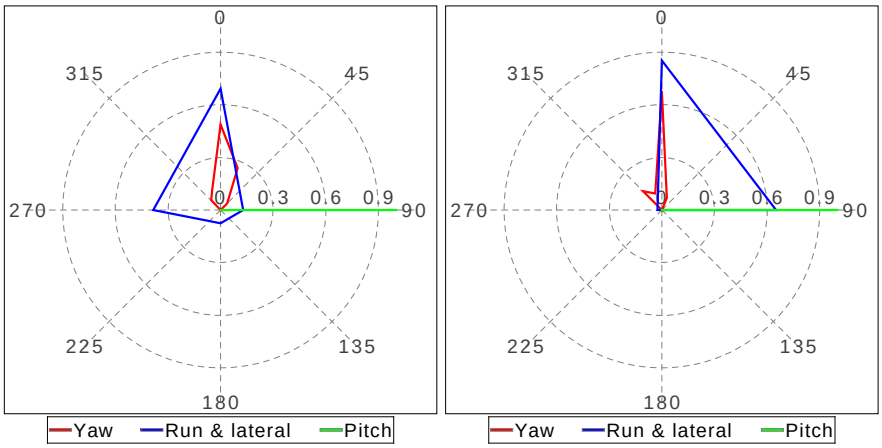


Figure 4.29: *Two distributions of movement actions for the same decision and same kind of stimuli but different values. The left graph is for a stimulus representing a weapon on the left of the agent and very close. The right graph is for a stimulus representing also a weapon but it is on the right of the agent and is also very close. The weapon is about the same height as the agent.*

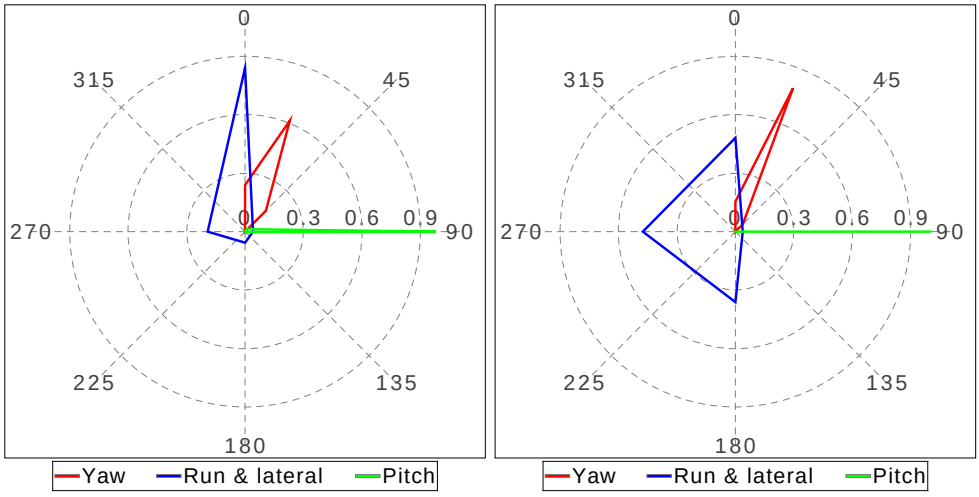


Figure 4.30: *Two distributions of movement actions for the same stimulus but different decisions. The two graphs are for stimulus representing an enemy player, right of the agent, moving to the right of the agent and at an average distance.*

All the learning do not gives good results. For example in the figure 4.31, the distributions are given for a navigation point at the left of the agent. In one decision (left graph), the agent will move toward the navigation point by moving laterally to the stimulus, but in an other, the agent will just go forward. We remind that each distribution gives the action to do for one stimulus, without taking the others into account. The second distribution is thus obviously "wrong" because it will make the agent go in a direction potentially harmful for the agent.

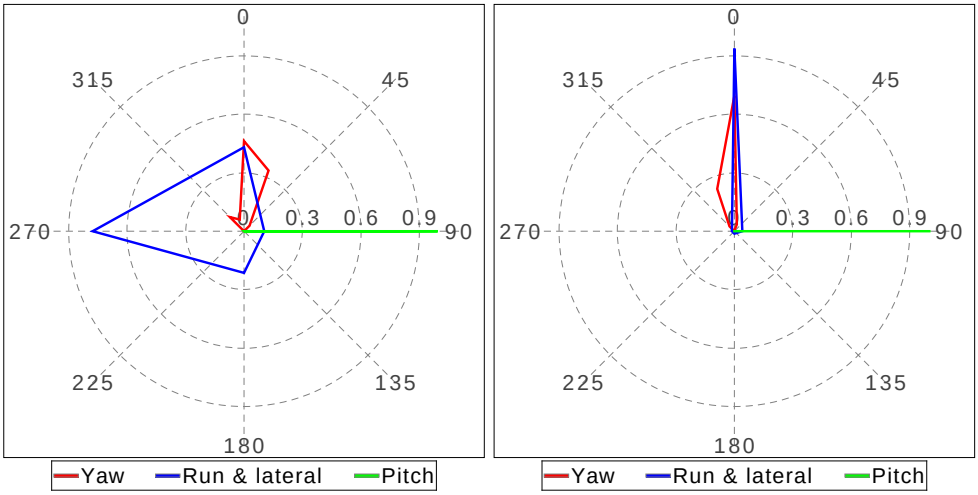


Figure 4.31: *Two distributions for the same stimulus but different decisions. The stimulus represent a navigation point on the left of the agent, same height and average distance.*

The study of the movement distributions shows that some knowledge is assimilated by the model parameters. The role of the decision becomes obvious with different tactics being used according to the state of the agent. However some distributions do not represent a believable behaviour at all. The reasons can be many: problem with reaction time, bad attention values, etc.

4.4.3.2 Signatures

In a previous study (Tencé and Buche, 2008), we presented a method to spot differences in the behaviour of players and agents. While it is not proved to be an indicator of believability, it may be used to spot non-believability. The idea is to monitor the movements of avatars and to extract some statistics. These statistics are the signatures of the behaviours, similarities can be spotted between agents and between players. The most important information is the differences spotted between players and agents which often reveal problem

in the behaviour of the agent. It is also possible to represent the distance between the signatures, visualizing the similarities in a more natural way.

In the following we will use three measures:

- Velocity relative to facing direction: angle between the velocity vector of the avatar and the facing direction. In figure 4.32 it is the angle between V^t and D^t.
- Direction change: angle between two consecutively monitored absolute facing direction. In figure 4.32 it is the angle between D^t and D^{t+1}.
- Velocity change: angle between two consecutively monitored absolute velocity. In figure 4.32 it is the angle between V^t and V^{t+1}.

For each of these angle, the number of time they have being monitored is counted and then is divided by the number of time the avatar has being monitored: a value of 1 means that the angle value was monitored 100% of the time, a value of 0.5 means 50%, etc. The frequency of the monitoring is 10Hz.

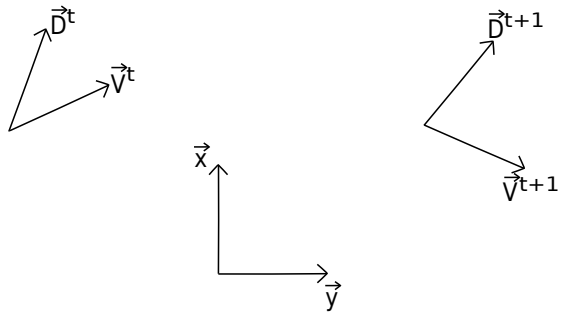


Figure 4.32: *Scheme of an avatar viewed from top at two following time steps t and $t+1$. $\overrightarrow{D}$ is the facing direction of the avatar which is also the direction in which the avatar aims and $\overrightarrow{V}$ is the velocity vector.*

We will compare the signatures for a player, a Chameleon agent, a Le Hy agent and an original UT2004 agent. The idea is to spot differences which represent noticeable differences in the behaviour, the objective being to be as close as possible to the player's signatures.

For the first signature (figure 4.33, top), we can see that for 0°, the UT2004 is the closest to the player, Le Hy and Chameleon are not moving forward enough. The player has a tendency to move laterally more to the right than to the left. Because Chameleon is the only model which parameters were

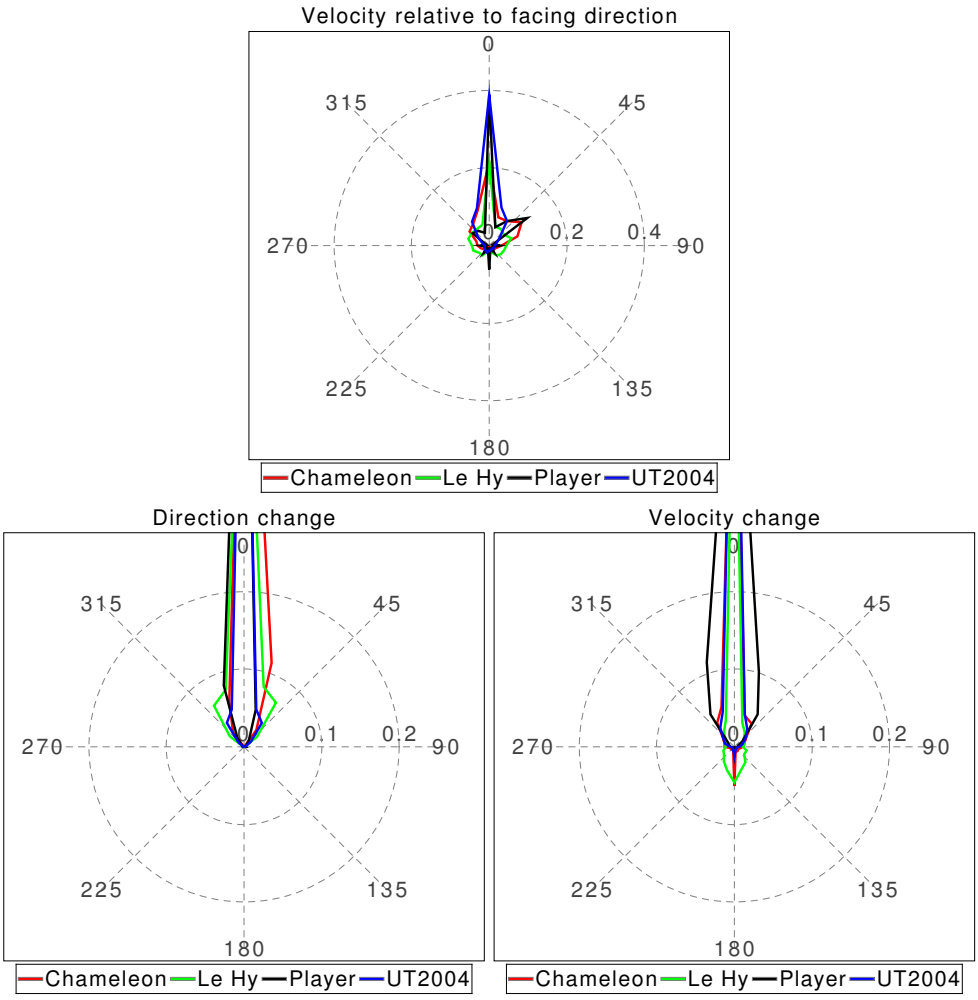


Figure 4.33: *Signatures (Tencé and Buche, 2008) of a Chameleon agent, a Le Hy's agent, a player and an original UT2004 agent.*

learned by imitation, it is the only one that reflect this habit. All the agents do not move enough backward. Finally, Le Hy's model seems to move too much backward and laterally at the same time compared to the player. The second signature (figure 4.33, bottom left) shows much less differences. Le Hy's model is making too much sudden turns as the UT2004 agent, in a lesser extent. Despite the imitation learning, Chameleon have tendencies to turn more right than left whereas the player turn slightly more to the left. The last signature (figure 4.33, bottom right) shows that all the agents are far from the player when it comes to to change in the velocity vector. The player seems to change very often the direction of the lateral movements while the other agents move less. Le Hy's agent seems to often change from forward to

backward direction, maybe because of the problem of the FEC but Chameleon suffers from the same problem, to a lesser extent.

In order to better visualize the differences between the signatures, we measured the distance between each signature with the Earth Mover's Distance (EMD). This distance allows the measure to take into account that turning 80° right and 90° right is almost the same but 90° right and 90° left is very different. So as to visualise the distance we represent the signatures on a plane using the Multidimensional Scaling (MDS) method. Both the EMD and MDS are detailed in (Tencé and Buche, 2008). The idea is that close (Euclidean distance) representations of the signatures in the MDS are close in in the EMD distance and thus are similar, the contrary being also true. As it is only a question of relative distance to the other, the graphs do not have any tics or values on the axis. In order to have a more complete study of the signatures we represent the signatures for the Chameleon and the Le Hy agent as well as seven different players and nine UT2004 agents, each with a different skill level.

For the first graph (figure 4.34, top), the UT2004 agent are widespread. The medium skilled ones are close to the players. Le Hy agent is pretty far from the players and Chameleon is not very far but not in the "cloud" of the players. That means it may be taken for a player but not an average one. In the second graph (figure 4.34, bottom left), the players are quite widespread and the UT2004 agents are still quite close. Le Hy agent is very far from the players and Chameleon is a bit far also. Finally, for the last graph (figure 4.34, bottom right), Le Hy agent is very far from the players, Chameleon is quite far and the agents are not very far but can be clearly separated from the players. To conclude, it appears that some UT2004 agents are the closest to the players, then comes Chameleon and then Le Hy's agent. The fact the agent from the game are this close is because character designers used a lot of time improving the way their agent moves. However, it may not apply to the behaviours.

It seems that Chameleon managed to perform better than Le Hy's agent. Because the discretization of the actions and the interface between the game and the model can be greatly improved, UT2004 agents perform better than Chameleon. As all the details of the agent's behaviour are important, this should be improved on Chameleon in order to be able to fool the players into thinking Chameleon is an other player.

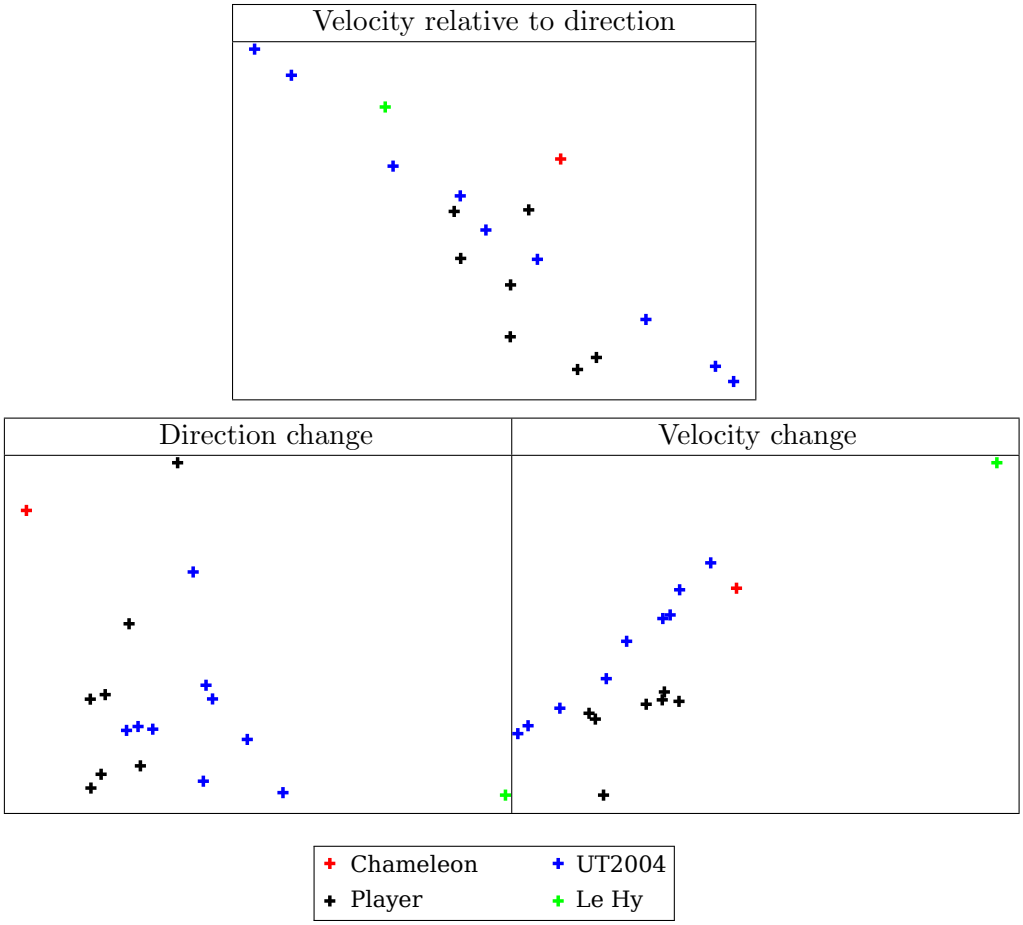


Figure 4.34: *Earth Mover's Distance (EMD) between the signatures represented using the Multidimensional Scaling (MDS) for one Chameleon, one Le Hy agent, seven different players and nine UT2004 agent, each one with a different skill level. The correlation factors for the MDS for all the representations are very high* (> 0.98).

4.4.3.3 Believability

We did not make a complete believability evaluation because, as the previous results suggest, the quality of the behaviours of Chameleon are not worth the effort. Indeed, a believability evaluation mainly consist in making a large number of people give their feeling about the agent. Gathering enough people and making them assess the believability of agents is very time-consuming. We will rather list believable and non-believable behaviours according to our knowledge of the game and players.

Believable Behaviours We noticed the following learned believable behaviours:

- Aims at an enemy when seeing it [B1: Reaction]
- Shoot mainly when enemy is in the agent's sight [B1: Reaction]
- Explore the environment [B5: Understandable]
- Changes its weapon for the best one [B1: Reaction]
- Stands still for some seconds mimicking reflection [B5: Understandable]
- Is truly unpredictable [B4: Unpredictability] and [B3: Variability]
- Evolve during learning in a noticeable way [B9: Evolution] and [B10: Fast Evolution]

Knowing that the agent starts with little knowledge of the environment and its rules, the learning seems to be pretty efficient. Basic associations are made (enemy⇔shoot) as well as more complex behaviour like moving in a maze-like environment. The probability play well its role even if the raw parameters make the movement of the agent choppy. The behaviour is much more smooth when a power (between 5 and 10) is applied to the distributions before the random pick is done.

Non-Believable Behaviours However, we noticed also the following learned non-believable behaviours:

- Does not aim accurately at the enemies [B1: Reaction]
- Often becomes stuck for several seconds against walls [B1: Reaction]
- Seems to pick weapons on the ground fortuitously [B5: Understandable]
- Forget information which is not any more in the FOV of the agent [B8: Memory]
- Does not plan [B7: Planning]

There are several potential explanation for these problems. First the agent may have insufficiently accurate motors and sensors to aim correctly at the enemies. The tendency to get stuck may come from incorrect attention values. As showed in section 4.4.3.1, some distributions are not valid which may come from incorrect updates due to a too high attention or incorrect reaction time. The fact the agent seems to pick fortuitously weapons on the ground may come from the same problem. It may also be related to the problem of exaggeration: agent have to overdo for us to understand its behaviour [B5: Understandable]. Finally, for the memory problem, it shows that the decision mechanism is not sufficient to give the illusion of memory [B8: Memory]. More complex sensory informations may be useful like for example a probability distribution of the position of the enemies knowing its last position.

The EM algorithm makes the agent behave more like a player than before the learning. The base of the behaviours are learned but they lack in efficiency to be considered as believable. Some solutions may improve the results and will be discussed further in the next chapter.

Conclusion of 4.4

In this section we showed that the EM can handle all the observations of a teacher without delay. during the learning the log-likelihood increase as expected but some distributions converge better than others. The fact attention function are not learned seems to cause problems in both the convergence and the resulting behaviours. The merging of the parameters allows the agent to generalise over several sequences of observations even if some improvement may be needed. Finally, the behaviour of the agent proves that it indeed learned part of the teacher's behaviours but it is not enough to make the agent believable.

Conclusion of 4

In this chapter we presented the results of four modifications to Le Hy's model in order to answer better to the believability requirements listed in section 1.2.2. The semantic refinement allowed to reduce the number of parameters in order to increase the speed of learning. The attention functions allowed to fix the problems of the FEC and IP. The GNG makes the agent able to adapt to unknown environments and to use a representation of the environment consistent with its use by the players. Finally the EM makes the agent learn almost all the behaviours by observing a teacher. This learning is fast allowing the agent to reproduce some of the habits of the teacher. However, there are still some unlearned or mis-learned behaviours which make Chameleon agent non-believable. The reason are many and will by discussed in the next chapter.

Contents of Chapter 5

Chapter 5

Conclusion

Summary of 5

The believability of characters in video games and more generally in virtual environments plays a role in the feeling of presence these environments can provoke (see chapter 1). According to chapter 2, one of the behaviour model the most promising to control believable characters is Le Hy's model. We proposed four enhancements or replacements to some parts of the model in chapter 3: a semantic refinement, an attention selection mechanism, an algorithm to learn the topology of the environment and finally a revamped Expectation-Maximization algorithm (EM) to learn the parameters of the model. In chapter 4 we found that our propositions indeed improved the results but that agents using this model are still far from being believable. We propose some future improvements: variable reaction time for the learning, update of the attention functions in the EM, improvements in the merging of the parameters after learning, improving the discretization of the stimuli and actions by learning, etc.

Introduction of 5

In this thesis we proposed a general model for the control of believable characters for virtual environments and more specifically video games. We also wanted this model to be able to adapt to new environments and new rules without too much work from the character designers. We hope that by providing such model, video games will be more vivid, appealing and more entertaining to the players. Of course, as believability is very hard to achieve, we relied on previous models and try to enhance them to achieve better results.

Plan of 5

In this chapter we will first remind the different requirements for agents to behave in a believable manner (section 5.1). Then we link these requirements to the solutions and the propositions we made in section 5.2. However the proposed model does not fulfil all the requirements (section 5.3) and may not appear as being believable to players. We give possible ways to correct these problems in order to make the agent fulfil all the requirements and to be able to measure the final result in an accurate manner (section 5.4).

5.1 Bottleneck

The general goal of this thesis is to identify and find solutions to resolve some of the bottlenecks impeding the creation of virtual environment where the feeling of presence is strong. We focus on the making of believable characters in video games which is already a challenge by itself. We detail what are the problems to be resolved to make a character believable.

5.1.1 Objectives

First the definition of believability is subject to debate. In this thesis we choose, as other researchers, the following definition:

> A believable agent is computer program able to control an avatar in a virtual environment so that other human users in the environment think the avatar is controlled by another human user.

The idea is to make human users feel there are surrounded by intelligent sentient beings.

Of course we do not want the agent to be specific to a video game but to be flexible enough to adapt to new environments and rules without too much work from programmers. This objective makes the modelling of believable agent even more difficult.

5.1.2 Requirements for Believability

According to other researchers' studies and our own, we listed several requirements a believable agent must fulfil (see table 5.1). The direct goal of this thesis is to find a behaviour model for an agent which fulfils these 10 requirements.

Each of these requirements includes a series of problems to be resolved. The agent must react to events in the environment [B1: Reaction] but it should react in a coherent manner. The time it takes to react [B2: Reaction time] is also very complicated to simulate as it depends on the complexity of the

Believability requirement	Summary of the requirement
[B1: Reaction]	React to the players and changes in the environment
[B2: Reaction time]	Simulate a human-like reaction time
[B3: Variability]	Have some variability in the actions
[B4: Unpredictability]	Surprise the players with unpredictable behaviour
[B5: Understandable]	Have a understandable behaviour
[B6: Perception]	Have human-like perception
[B7: Planning]	Plan actions in the future to avoid mistakes
[B8: Memory]	Memorize information
[B9: Evolution]	Evolve to avoid repeating mistakes
[B10: Fast Evolution]	Evolve fast enough for the players to see it

Table 5.1: *List of the requirements for a character to be believable.*

information but as this may be very small variations, does the players notice them all? The agent must have some imperfections [B3: Variability] and must avoid repetitiveness [B4: Unpredictability] but too much randomness can be very harmful to the believability. One of the bigger defy is to make the behaviour of the agent understandable to the players [B5: Understandable] so that they can guess the intentions of the agent. However the agent must not appear too "dumb" to be challenging. The technically biggest challenge is to make the agent have human-like perception [B6: Perception]. The best solution would be to have the agent use the rendering of the 3D environment with colour, shadows, etc. The problem is that it would require the agent to use techniques from computer vision which are not advanced enough for the agent to have all the information it needs. Instead we choose to give the agent the perception of the avatar which are an acceptable approximation. The agent have to take into account time with memories from the past [B8: Memory] and planning in the future [B7: Planning]. As the virtual environment are often complex it may rapidly become intractable to simulate all this mechanisms accurately. Finally, the agent has to be able to evolve [B9: Evolution] for the players to notice its ability to learn [B10: Fast Evolution] and also because the agent must be able to adapt to a whole range of virtual environments.

5.2 Contributions

In order to best fulfil the requirements, we first identified Le Hy's model (Le Hy et al., 2004) as a good base to develop a model for believable characters in video games. As it links sensors to actions, it can fulfil [B1: Reaction]. The quality of the responses to stimuli is determined by the parameters. the model is probabilistic so it can simulate very well randomness needed for [B3:

Variability] and [B4: Unpredictability]. As the model is based on the HMM theory, the model can handle dynamics like reaction time [B2: Reaction time] and party memory [B8: Memory]. Le Hy developed learning algorithms so that the model can evolve rapidly by imitating players [B9: Evolution] [B10: Fast Evolution].

The work in this thesis aim at improving the model parameters to better the quality of the responses to stimuli [B1: Reaction]. It also tries to make the agent handle the perceptions better [B6: Perception] and in a more human-like fashion which will also make the behaviour of the agent easier to understand for the players [B5: Understandable]. Finally, work has been done to allow the agent to adapt to a wide range of different environments very rapidly [B9: Evolution] [B10: Fast Evolution]. This evolution has a very important impact on the final result as it define the model parameters, the probability distributions, which impact the quality of the reactions [B1: Reaction], the randomness of the behaviours [B3: Variability] [B4: Unpredictability], the dynamics in the behaviours [B2: Reaction time] [B8: Memory] and if the behaviour is exaggerated or not [B5: Understandable].

5.2.1 Semantic Refinement

The goal of the semantic refinement is to reduce the number of parameters by defining *a priori* some independence between random variables. By reducing the number of parameters and giving the model some knowledge, the learning should be faster without reducing the expressiveness of the model.

The principle of the proposition is the following: instead of considering all the stimuli for each choice (decision and actions), each choice uses stimuli with different level of granularity. Global and spatially inaccurate stimuli, called high-level stimuli, are used for the choice of the decisions and also for actions which only involve the agent. Spatially accurate stimuli, called low-level stimuli, are used to perform actions which aim at interaction with the environment.

This proposition allowed a significant reduction of the number of parameters (around 85%). On top of allowing a faster evolution of the model [B10: Fast Evolution], it also makes possible for character designers to modify the model parameters. Indeed, as the number of parameters in much lower, the task is less tedious.

5.2.2 Attention Selection Mechanism

The goal of the attention selection mechanism is firstly to break the complexity of the model by avoiding the combinatorial explosion of the stimuli.

Indeed, we cannot express the actions to make for each combination of all possible stimuli. By avoiding this complexity, the number of parameters will be reduced allowing an even faster learning. Finally the idea is to find a way to break the complexity without reducing too much the expressiveness of the model and by making the agent simulate the way humans handle high quantity of information.

The idea is similar to the methods developed in Le Hy: instead of specifying actions for all the possible combination of all stimuli, the actions are specified for one unique stimuli. The trick is then to find a way to merge all these distributions into one when the agent is facing an environment composed of several stimuli. Our mechanism is very simple: at each time step the agent is focusing on one high-level stimulus and one low-level stimulus. The choice of the stimuli to focus on is based on attention functions: the higher the value for a stimulus, the higher the chances this stimulus will be picked if the agent perceive it.

The attention selection mechanism allows a very important reduction of the number of parameters by dividing the number of parameters by 10^5 in our application. It also seems to fix issues which affects Le Hy's methods. Indeed, the hypothesis for the application of the Inverse Programming (IP) is usually not met in applications with high number of stimuli and the Fusion by Enhanced Coherence (FEC) makes the agent constantly switch between stimuli. The model parameters can then be learned faster [B10: Fast Evolution]. On top of that, the attention selection mechanism simulate, in a very simple way, how human handle high quantity of information. This method can then make the agent perceive more like a human would do [B6: Perception]. Finally, by making the agent focus mainly on very important stimuli, we can make the agent behave in a more understandable way for the players: the agent will act as if there were only one stimulus visible to it, making its behaviour and intention very clear [B5: Understandable].

5.2.3 Learning of the Representation of the Environment

In order to make the agent adapt to unknown environments without help from programmers, we wanted our agent to learn by imitation the layout of the environment. As our goal is believability, we want also the representation to reflect how players use the environment to make it easier for the agent to reproduce these behaviours.

In order to represent and to learn the environment, we modified a model named Growing Neural Gas (GNG) which updates a graph according to input coordinates. The graph stretch and grow to cover the whole space where the player has been monitored to go. The model has been modified to be able to

learn continuously on a player without growing indefinitely but being able to grow if the teacher begins to use a new part of the environment.

The representation learned is very similar to usual representations found in video games which are used by the agents to move in the environment. The agent is thus able to adapt to totally unknown environments [B9: Evolution]. The learning is very fast as it takes less than 10 minutes with several teachers [B10: Fast Evolution]. However, whether it helps or not the agent to behave in a believable way has not been assessed.

5.2.4 Expectation-Maximization algorithm (EM) for the Learning of the Model Parameters

The objective of the EM is to replace the learning algorithm in Le Hy's work by a unique algorithm which avoid simplifications. The learning will be more demanding in computing power but the learned behaviours can be more complex. As most of the behaviour of the agent depends on the parameters and thus on the learning, this proposition is crucial for the final behaviour.

In order to avoid making simplification hypothesis and to adjust the learning algorithm to our need, we applied an EM technique from scratch. The likelihood function, which gives the probability for the model to generate given observations, has been expressed and each part of the model can be optimized. We proposed some update rules to optimize the distributions of the model to increase the value of the likelihood function. However, due to the complexity of the attention functions, we did not manage yet to find a good optimization technique. As the EM is applied to short sequences we proposed a way to merge all the results into a global one by updating these global parameters after with each result of the learnings.

The results given by the learning algorithm are mitigated. Some associations are well learned, like for instance attacking when an enemy is in sight and moving toward free places in the environment [B9: Evolution] [B1: Reaction]. It also learns to delay some reactions [B2: Reaction time]. However, as some associations are not learned or mis-learned, the global behaviour is not believable. The tests of an implementation of the algorithm showed that the learning is very fast, with distributions leaned in less than 40 minutes, and can be done without loosing information and without too much delay [B10: Fast Evolution].

As a conclusion, the propositions allowed the agents controlled by our model to fulfil more of the requirements listed in the beginning. The model and the learning algorithm make the agent react to stimuli often in a human-like

fashion [B1: Reaction] and can simulated delay in the reaction [B2: Reaction time]. The probabilities allow the agent to fulfil [B3: Variability] and [B4: Unpredictability]. However, learned distributions can make the movements choppy, so it can be necessary to modify the way the model pick decisions. The agent can adapt to new environments and rules with the GNG and the EM [B9: Evolution] with results converging very rapidly [B10: Fast Evolution]. The attention selection mechanism makes the behaviour of the agent reflect more human-like perceptions [B6: Perception] and more understandable for players [B5: Understandable]. Finally, the agent can "remember" as its choice of decision is based on the previous one [B8: Memory]. This information is however very limited. Indeed, it often does not give enough information for the agent to react to stimuli which are not visible any more.

5.3 Limitations

All the propositions we presented in this thesis did not answer all the requirements or fulfil them only partly. Because of this we chose not to perform a full believability evaluation until these problems are fixed. We will list the most obvious unbelievable behaviours and associate them with the requirements.

5.3.1 Low Actions Accuracy

Our implementation of the Chameleon model have difficulties with performing accurate actions like for example aiming or walking on narrow paths. Of course, the agent has to perform actions with a certain degree of error [B3: Variability] to simulate the inaccuracy of the human actions. However, the actions of our agent are sometimes so inaccurate that it breaks the illusion of believability.

The problem mainly comes from the interface between the model and the video game. The discretisation of the stimuli and the actions, necessary to define the discrete distributions are not fine enough for the agent to use them efficiently. As a result the model cannot fulfil entirely [B1: Reaction] and [B6: Perception].

5.3.2 Incorrect Learned Associations Between Stimuli and Actions

The agent we coded despite being able to learn the base behaviour to explore the environment still becomes stuck against walls. The agent goes straight to a wall and continues walking toward it even when it collides with it. This is a major problem as it is clearly identifiable as a non-human behaviour by

other players. It also impedes the agent to play the game normally, exploring, hunting and grabbing important items.

Even when the agent explores the environment, it should modify its course to grab items on the ground, mostly weapons. This allows the agent to be more powerful and to beat the other players. It appears to observers that this behaviour it not well learned because the agent often miss the items of may sometimes avoid them even it the agent needs the item.

There are several explanations for these problems. First, we approximated the teacher's reaction time by a constant which is a very simple assumption. It is possible that wrong associations are learned because the learning algorithm does not have the right data. The second possible explanation is that the attention functions are not learned and thus their values are not correct. The learning algorithm may have considered that the teacher focused on an insignificant stimulus. Finally, the merging of all the results of the EM may not give satisfactory distributions. It is most probable that all the three explanations combined have a negative impact on the final result making the agent unable to act correctly [B1: Reaction] and coherently [B5: Understandable] and hinder its evolution abilities [B9: Evolution].

5.3.3 Incomplete Memory

When playing against the agent, it rapidly becomes obvious that it lacks of memory mechanisms: when an information is not any more in the Field Of View (FOV) of the agent, the agent acts as it never existed. As the agent have human-like perceptions and thus a quite narrow FOV (90° horizontally and 67.5° vertically to correspond to what is rendered on a player's screen), the problem happens quite often.

The reason of this problem is obvious: the model is designed so that the model has a memory of the previously taken decision, but does not directly consider previous stimuli. This method is too limited for the agent to simulate a believable memory [B8: Memory].

5.3.4 Lack of Planning Mechanisms

The agent does not give the illusion of having any long and mid-term goals [B7: Planning]. However, when given information on the speed of other players, the agent can learn to anticipate their movements and thus giving the impression that the agent can plan some seconds ahead. The fact the agent cannot plan further makes it easy to outsmart it and thus breaks the illusion of believability.

The reason why the agent cannot plan further is because it does not have any mechanism for that in the model. Planning can be very complicated in complex environments but some tools could be given to the agent to at least give the illusion of planning.

To sum up, the Chameleon model in its current version cannot make other player believe that it is a player for long because of the interface with the games [B6: Perception] [B1: Reaction], problems with learning [B5: Understandable] [B2: Reaction time] and lack of mechanisms to memorize [B8: Memory] and to plan [B7: Planning].

5.4 Future Work

The limits we pointed out can, of course, be fixed but the solutions can be difficult to implement. We give some ideas of the principle improvements that can be added to the model.

5.4.1 Extension of the Attention Selection Mechanism

The attention selection mechanism, despite its simplicity, can give pretty believable results. The problem is that players can focus on several stimuli at one time. The idea is to add new attention functions. Instead of making the agent focus only on one high and one low-level stimulus, the agent can focus on one stimuli for each group of action. For instance, the agent can focus on one high-level stimulus for the choice of the decision, one high-level stimulus for the reflexive actions, one low-level stimulus for the movements and one low-level stimulus for the interactions. The agent could then move trough a door and turn on the light by focusing on two different stimuli: the navigation point at the door and the light button next to the door.

This improvement makes also possible the involvement of the decision in the choice of the high-level stimulus for reflexive action. Indeed, as the decision can be chosen prior to the choice of the high-level stimulus for the reflexive action, it can be taken into account, making the whole behaviour more complex.

5.4.2 Improvements on the Learning Algorithm

The main problem the learning algorithm suffers at the moment is the influence of the teacher's reaction time on the observations. As unrelated stimuli-actions examples are given to the algorithm, incorrect distributions are learnt. A mechanism should be found to align the related stimuli and actions while the EM converges.

An other issue is that the actual version of the learning algorithm is incomplete because the attention functions are not learnt. These functions are essential for the agent because they have an important impact on both the learning of the other distributions and how the agent behaves. A very simple case of study should be designed to fully understand the influence between the attention functions and the other distributions during the learning.

Finally, the merging of the final results of the EM should also be studied into detail to find why the global parameters are not the best initialisers for the learning algorithm. Maybe an other mechanism could be used to associate the decisions, which meaning are unknown, between two results of the EM.

5.4.3 Discretisation of the Stimuli and Actions

We saw that the interface between the model and the video game has an important impact on the perceived behaviour of the agent. The discretisation of the stimuli and actions have to be done very carefully to avoid having too much parameters but still allowing the agent to be accurate. Learning the discretisation would make the model even more flexible, making its adaptation to new environments very easy.

It can also be possible to remove the discretisation and to switch to a full continuous distribution model. The main problem with this improvement is to find a compact representation for the distribution, for instance Gaussian mixtures, and to find a fast and efficient algorithm to learn these distributions.

5.4.4 Planning

The model lacks of a mechanism to make plans in the future. We saw that with the EM estimators it is possible for the model to find out the teacher's intentions. By integrating the estimator into the model itself, it may be able to better understand the other players and try to think ahead. It can be also possible to use the stimuli as a base for planning. They could be associated with a probability of presence, decreasing from the last seen position.

Finally we used only a game mode where all the players play for themselves and the objective of the game is really simple. By making the agent play team games where each player has a role and the whole team has to have a strategy, the challenge would be more interesting. The model would certainly need some adjustments so as the agent is able to be integrated seamlessly in a team of human players.

5.4.5 Believability Evaluation

Once the main improvements have been done to the model, a thorough evaluation of the believability of the Chameleon model, Le Hy's model, the original UT2004 agents and the players should be done. An overview of the evaluation methods as already been done in (Tencé et al., 2010). The actual choices of the implementation of this evaluation should be carefully studied with psychologists. Indeed, believability experiments must involve humans to express their feelings about the test subjects.

Bibliography

Anderson, J. (1993). *Rules of the mind*. Lawrence Erlbaum Associates. xii, 22

Angluin, D. (1982). Inference of reversible languages. *Journal of the ACM (JACM)*, 29(3):741–765. 21

Angluin, D. (1992). Computational learning theory: survey and selected bibliography. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 351–369. ACM. 28

Artieres, T., Marukatat, S., and Gallinari, P. (2007). Online handwritten shape recognition using segmental hidden markov models. *IEEE transactions on pattern analysis and machine intelligence*, 29:205–217. 24

Bates, J. (1992). The nature of characters in interactive worlds and the Oz project. Technical Report CMU-CS-92-200, School of Computer Science, Carnegie Mellon University. 4, 6, 22

Bauckhage, C., Gorman, B., Thurau, C., and Humphrys, M. (2007). Learning Human Behavior from Analyzing Activities in Virtual Environments. *MMI-Interaktiv*, 12:3–17. 24, 25, 32

Bauckhage, C. and Thurau, C. (2004). Towards a Fair'n Square Aimbot–Using Mixtures of Experts to Learn Context Aware Weapon Handling. In *Proc. GAME-ON*, pages 20–24. 32

Baum, L., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171. 42

Bengio, Y. and Frasconi, P. (1995). An input output HMM architecture. In *Advances in neural information processing systems*, pages 427–434. Citeseer. 35

Biernacki, C., Celeux, G., and Govaert, G. (2003). Choosing starting values for the em algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Computational Statistics & Data Analysis*, 41(3-4):561–575. 88

Blackmore, S. (1999). *The meme machine*. Oxford University Press, USA. 30

Bozinovic, R. and Srihari, S. (1982). A string correction algorithm for cursive script recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):655–663. 24

Bryant, B. and Miikkulainen, R. (2006). Evolving stochastic controller networks for intelligent game agents. In *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*, pages 3752–3759. 9

Burdea, G. C. and Coiffet, P. (2003). *Virtual Reality Technology*. John Wiley & Sons, Inc., New York, NY, USA, 2 edition. 2

Burkert, O., Kadlec, R., Gemrot, J., Bida, M., Havlicek, J., Dorfler, M., and Brom, C. (2007). Towards Fast Prototyping of IVAs Behavior: Pogamut 2. In *Intelligent Virtual Agents*, volume 4722, pages 362–363. Springer. 45

Calinon, S. and Billard, A. (2007). Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 255–262. ACM. xii, 24, 31

Cass, S. (2002). Mind games. *IEEE Spectrum*, 39(12):40–44. 10, 19

Cavazza, M., Charles, F., and Mead, S. (2002). Character-based interactive storytelling. *IEEE Intelligent Systems,*, pages 17–24. 4

Charles, D. and McGlinchey, S. (2004). The past, present and future of artificial neural networks in digital games. In *Proceedings of the 5th international conference on computer games: artificial intelligence, design and education. The University of Wolverhampton*, pages 163–169. Citeseer. 18

Crawford, C. (2004). *Chris Crawford on Interactive Storytelling*. New Riders Games. 4

Crouse, M., Nowak, R., and Baraniuk, R. (1998). Wavelet-based statistical signal processing using hidden markov models. *Signal Processing, IEEE Transactions on*, 46(4):886–902. 34

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38. 74

Donikian, S. (2001). Hpts: a behaviour modelling language for autonomous agents. In *Proceedings of the fifth international conference on Autonomous agents*, pages 401–408. ACM. 19

Dromey, R. (2003). From requirements to design: Formalizing the key steps. In *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on*, pages 2–11. IEEE. 19

Feldman, J. and Ballard, D. (1982). Connectionist models and their properties. *Cognitive science*, 6(3):205–254. 17

Fikes, R. and Nilsson, N. (1971). Strips: a new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3/4):189–208. 22

Florez-Larrahondo, G. (2005). *Incremental learning of discrete hidden Markov models*. PhD thesis, Mississippi State University. 33, 41, 43, 44, 74

Fritzke, B. (1995). A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press. xiv, 70

Gibson, J. (1986). *The ecological approach to visual perception*. Lawrence Erlbaum Associates. 2

Glass, J., Chang, J., and McCandless, M. (1996). A probabilistic framework for feature-based speech recognition. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 4, pages 2277–2280. IEEE. 24

Gorman, B. and Humphrys, M. (2007). Imitative learning of combat behaviours in first-person computer games. In *Proceedings of CGAMES 2007, the 11th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games*. 10, 18, 32, 33

Gorman, B., Thurau, C., Bauckhage, C., and Humphrys, M. (2006a). Bayesian imitation of human behavior in interactive computer games. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)-Volume 01*, pages 1244–1247. IEEE Computer Society Washington, DC, USA. 24, 25

Gorman, B., Thurau, C., Bauckhage, C., and Humphrys, M. (2006b). Believability testing and bayesian imitation in interactive computer games. In *From Animals to Animats 9*, volume 4095, pages 655–666. Springer. 25, 32

Heeter, C. (1992). Being there: The subjective experience of presence. *Presence: Teleoperators and virtual environments*, 1(2):262–271. 2

Held, R. and Durlach, N. (1991). Telepresence, time delay and adaptation. *Pictorial communication in virtual and real environments*, pages 232–246. 2

Hinton, G. (1989). Connectionist learning procedures. *Artificial intelligence*, 40(1-3):185–234. 18

Hoffman, H. G., Richards, T., D, P., D, P., Coda, B., Richards, A., and Sharar, S. R. (2003). The illusion of presence in immersive virtual reality during an fmri brain scan. *Cyberpsychology & Behavior*, 6:127–131. xi, 3

Isla, D. (2005). Handling complexity in the halo 2 ai. In *Game Developers Conference*. 9, 17

Johnson, D. and Wiles, J. (2001). Computer games with intelligence. In *Fuzzy Systems, 2001. The 10th IEEE International Conference on*, volume 3, pages 1355–1358. IEEE. 18, 19

Krogh, A., Larsson, B., Von Heijne, G., and Sonnhammer, E. (2001). Predicting transmembrane protein topology with a hidden markov model: application to complete genomes1. *Journal of molecular biology*, 305(3):567–580. 34

Laird, J. and Duchi, J. (2000). Creating human-like synthetic characters with multiple skill levels: a case study using the Soar Quakebot. In *Simulating Human Agents, Papers from the 2000 AAAI Fall Symposium*, pages 75–79. 9

Laird, J. and Lent, M. (2001). Human-level ai's killer application: Interactive computer games. *AI Magazine*, 22(2):15–26. 4, 5, 10

Laird, J., Rosenbloom, P., and Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine learning*, 1(1):11–46. 22

Le Hy, R. (2007). *Programmation et apprentissage bayésien de comportements pour des personnages synthétiques, application aux personnages de jeux vidéos*. PhD thesis, Institut national polytechnique de Grenoble. vi, xi, xii, 20, 38, 39, 41, 48, 51, 57, 94, 132, 166

Le Hy, R., Arrigoni, A., Bessière, P., and Lebeltel, O. (2004). Teaching bayesian behaviours to video game characters. *Robotics and Autonomous Systems*, 47(2-3):177–185. xiii, 12, 15, 16, 24, 25, 32, 33, 34, 38, 40, 46, 57, 145

Levinson, S. (1983). An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *The Bell System Technical Journal*. 24

Livingstone, D. (2006). Turing's test and believable AI in games. *Computers in Entertainment*, 4(1):6. 7, 9, 10

Loomis, J. (1992). Distal attribution and presence. *Presence: Teleoperators and Virtual Environments*, 1(1):113–119. 2

Loyall, A. B. (1997). *Believable agents: building interactive personalities*. PhD thesis, Carnegie Mellon University. 4, 6, 9, 10, 22

Mac Namee, B. (2004). *Proactive persistent agents: using situational intelligence to create support characters in character-centric computer games*. PhD thesis, University of Dublin. 5, 10

Mari, J., Fohr, D., and Junqua, J. (1996). A second-order HMM for high performance word and phoneme-based continuous speech recognition. In *icassp*, pages 435–438. IEEE. 24

McMahan, A. (2003). Immersion, engagement and presence. *The video game theory reader*, pages 67–86. 4

Meltzoff, A. and Moore, M. (1977). Imitation of facial and manual gestures by human neonates. *Science*, 198(4312):75–78. xii, 25, 30, 32

Mitchell, T. (1997). *Machine Learning*. 27

Murphy, K. (2002). *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley. xii, 35, 36, 37

Newell, A., Rosenbloom, P. S., and Laird, J. E. (1987). Soar: an architecture for general intelligence. *Artificial intelligence*, 33(1):1–64. 22

Orkin, J. (2006). Three States and a Plan: The AI of FEAR. In *Proceedings of the 2006 Game Developers Conference*. xix, 22, 23

Pinchbeck, D. (2008). Trigens can't swim: intelligence and intentionality in first person game worlds. In *Proceedings of The Philosophy of Computer Games 2008*. University of Potsdam. 9

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. 34, 42

Rao, R., Shon, A., and Meltzoff, A. (2004). A bayesian model of imitation in infants and robots. In Nehaniv, C. L. and Dautenhahn, K., editors, *Imitation and Social Learning in Robots, Humans, and Animals*, pages 217–248. Cambridge University Press. 25, 32

Reilly, W. (1996). *Believable social and emotional agents*. PhD thesis, University of Birmingham. 22

Riedl, M. and Stern, A. (2006). Believable agents and intelligent story adaptation for interactive storytelling. *Technologies for Interactive Digital Storytelling and Entertainment*, pages 1–12. 4

Riedl, M. and Young, R. (2005). An objective character believability evaluation procedure for multi-agent story generation systems. In *Intelligent Virtual Agents*, volume 3661, pages 278–291. Springer. 2, 6

Robert, G. and Guillot, A. (2005). A motivational architecture of action selection for non-player characters in dynamic environments. *International Journal of Intelligent Games & Simulation*, 4(1):1–12. 22

Sanchez, S., Luga, H., and Duthen, Y. (2006). Learning classifier systems and behavioural animation of virtual characters. In *Intelligent Virtual Agents*, pages 467–467. Springer. 22

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242. 30

Shepard, R. (1987). Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317. 58

Sigaud, O. and Wilson, S. (2007). Learning classifier systems: a survey. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 11(11):1065–1078. 22

Simmons, R. and Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1080–1087. 24

Slater, M., Linakis, V., Usoh, M., Kooper, R., and Street, G. (1996). Immersion, presence, and performance in virtual environments: An experiment with tri-dimensional chess. In *ACM Virtual Reality Software and Technology (VRST*, pages 163–172. 2

Slater, M., Usoh, M., and Steed, A. (1995). Taking steps: the influence of a walking technique on presence in virtual reality. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2(3):201–219. 2

Stanley, K., Bryant, B., and Miikkulainen, R. (2005). Evolving neural network agents in the nero video game. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*, pages 182–189. Citeseer. 28

Steuer, J. (1992). Defining virtual reality: dimensions determining telepresence. *Journal of Communication*, 42(4):73–93. 2

Tencé, F. and Buche, C. (2008). Automatable evaluation method oriented toward behaviour believability for video games. In Vicente Botti, Antonio Barella, C. C., editor, *GAME-ON 2008 9th International Conference on Intelligent Games and Simulation*, pages 39–43. EUROSIS. xvii, 28, 134, 136, 137

Tencé, F., Buche, C., De Loor, P., and Marc, O. (2010). The challenge of believability in video games: Definitions, agents models and imitation learning. In Mao, W. and Vermeersch, L., editors, *GAMEON-ASIA'2010, 2nd Asian Conference on Simulation and AI in Computer Games*, pages 38–45. Eurosis. 7, 50, 153

Thomas, F. and Johnston, O. (1981). *Disney animation: the illusion of life*. Abbeville Press. 6

Thrun, S. (1995). Learning to play the game of chess. *Advances in Neural Information Processing Systems*, pages 1069–1076. 28

Thurau, C., Bauckhage, C., and Sagerer, G. (2004). Learning human-like movement behavior for computer games. In *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04)*. 31, 32, 33, 69, 106

Thurau, C., Paczian, T., and Bauckhage, C. (2005). Is bayesian imitation learning the route to believable gamebots? In *GAMEON-NA'2005*, pages 3–9. 10, 30

Vinciarelli, A. (2002). A survey on off-line cursive script recognition. *Patter Recognition*, 35. 24

Weizenbaum, J. (1966). ELIZA – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45. xi, 10, 11

Wetzel, B. (2004). Step one: Document the problem. In *Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop, AAAI Press, Menlo Park, CA*, pages 11–15. 9

Wu, C. (1983). On the convergence properties of the em algorithm. *The Annals of Statistics*, pages 95–103. 74

Yamato, J., Ohya, J., and Ishii, K. (1992). Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 379–385. IEEE. 34

Zeng, Z., Goodman, R., and Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990. 21

Exploratory Research on Criteria Affecting Believability

Protocol

The experiment use two participants to evaluate the believability of an agent. We did the experiment 7 times, with a total of 14 different participants. We used the game UT2004 for the virtual environment with modified rules: the participant were asked to play a tag game.

The experience it done in two phases:

1. The two participants play one against the other during 5 minutes
2. Each participant plays against the evaluated agent during 5 minutes

Then the participant were given the following questionnaire, with no time limit to answer.

Questionnaire

Personal information	
Name:	First Name:

Experience in first person shooter games:
1 Never played/Rarely seen
2 Already played/seen
3 Played more than once per month
4 Played more than once per week
5 Played more than once per day

Which were the differences between the agent's and the player's behaviour?

Could you affirm that the character you saw was really artificial, and why?

Give the criteria giving the illusion an avatar is controlled by a human player:

What were your impressions on the experiment itself? (time, rules, etc.)

Abstract

This thesis aims at designing a behaviour model for the control of believable characters in video games. The character is controlled by a computer program we call an agent. We define a believable agent as a computer program able to control a virtual body in a virtual environment so that other human users in the environment think the virtual body is controlled by a human user. As more precise criteria are needed for the evaluation of the results, we define 10 requirements for a character to be believable, based on previous experiments and work: reaction, reaction time, variability, unpredictability, understandability, human-like perceptions, planning abilities, memory, evolution and fast evolution.

In order to fulfil these requirements, we studied the existing behaviour models developed both in the research and the industry. We grouped them into four types: connectionist models, state transition systems, production systems and probabilistic models, each one having its strengths and weaknesses. As one of the requirements is that the model is able to evolve, we had to find learning algorithms for the behaviour model. We find out that imitation is the best way to believability. With these studies in mind we find out that the behaviour model developed by Le Hy in his thesis (Le Hy, 2007, in French) answers to most of the requirements but has still some limitations.

We use an approach similar to Le Hy's but we made different choices in order to improve the believability. We first try to reduce the number of parameters in the model with a semantic refinement. Then we replace the two mechanisms to break the complexity of the probability distributions by an attention selection mechanism. This avoid the agent switching constantly between stimuli. We add to the model the ability to learn by imitation the layout of environments with a model named Growing Neural Network. Finally we totally revamp the learning algorithm with an Expectation-Maximization method.

The proposition makes the model able to learn how to act in the environment rapidly. Stimulus-action associations are made which make the agent look-like a human player. However the learning also makes wrong associations which destroy the illusion of believability. According to our studies, our model performs better than Le Hy's but work has still to be done on the model to achieve the final goal. We list in the conclusion the fulfilled requirements and potential solutions to fix the issues our model suffers.